Model Railroad System
A collection of utilities for Model Railroaders
Programming Guides

Robert Heller
Deepwoods Software
Wendell, MA, USA

January 22, 2015

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

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

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

Contents

Listings

List of Figures

List of Tables

Chapter 1

Introduction

This manual presents some information about how to write programs that use various parts of Model Railroad System to control and manage various aspects of operating your model railroad. The Model Railroad System includes code that interfaces with special hardware, including the PI Engineering Raildriver Console (see Chapter 3), a Bruce Chubb network of control nodes (see Chapter 5), and a Lenz XPressNet network (see Chapter 6). Also included are a collection of Tcl scripts that implement a collection of widgets that are useful for various aspects of programming utilities for a model railroad system.

Part I

Raildriver Programming (From C++ and Tcl)

Chapter 2

Direct access library for the Raildriver.

Coming soon.

Chapter 3

Writing clients for the Raildriver Server.

The PI Engineering Rail Driver Console is a working "model" of a modern locomotive control stand. It is equipped with a reverser, a throttle/dynamic brake lever, an automatic (trainline) brake, an independent (locomotive) brake, plus a collection of additional controls, buttons, and switches. This is a USB interface device. The Model Railroad System includes a hotplug daemon, written using libusb, that interfaces with this device. The hotplug subsystem launches this user-mode daemon, which connects to the Rail Driver Console through the USB interface and then listens on a TCP/IP port. Client programs can connect to the daemon through the TCP/IP port. Any number of client programs can connect and each can listen to different events. If more than one Rail Driver Console is plugged in, additional daemons are started and they will listen on different TCP/IP ports. The first daemon will listen on port 40990, and the second on port 41000 and the third on port 41010, and so on.

Command	**Description**
Exit	Disconnect from the server.
Clear	Clear event mask.
Mask maskbits	Add to the event mask.
Pollvalues maskbits	Get the current event state.
Led digits	Set the speedometer display
Speaker on / off	Turn the speaker on or off.

Table 3.1: Rail Driver Client Commands

Mask Name	Description
Reverser	Reverser lever event.
Throttle	Throttle / Dynamic brake lever.
Autobrake	Automatic (trainline) brake lever.
Independbrk	Independent (locomotive) brake lever.
Bailoff	Independent brake lever bailoff.
Headlight	Headlight switch.
Wiper	Wiper switch.
Digital1	Blue Buttons 1-8.
Digital2	Blue Buttons 9-16.
Digital3	Blue Buttons 17-24.
Digital4	Blue Buttons 25-28, Zoom up, Zoom down, Pan up, and Pan right.
Digital5	Pan down, Pan left, Range up, Range down, Emergency brake up, Emergency brake down, Alert, and Sand.
Digital6	Pantograph, Bell, Whistle up, and Whistle down.

Table 3.2: Rail Driver Maskbits Commands

Code	Message	Description
201	OK	Generic acknowledgment.
202	Events: maskbit [=(bits)]	One or more masked events.
299	GOODBYE	Server is about to close the connection. Sent in response to an Exit command.
502	Parse error	There was an error parsing a command.
503	message from the parser	A detailed message from the parser.
504	message: object 'something'	A detailed message from the parser.

Table 3.3: Rail Driver Server Messages

The client and server communicate by sending messages as ASCII text lines. The defined client messages are shown in Table 3.1. The maskbits define the events to listen for or to poll. They are listed in Table 3.2. The server sends messages prefixed with a three digit number. The first digit is a severity level. A severity level of 2 is informational and a severity level of 5 is error. The full set of messages sent by the server are listed in Table 3.3.

3.1 Sample Tcl Client Script

Listing 3.1: LocoTest.tcl

```
1  #!/usr/bin/tclsh8.6
   #*
   #* ------------------------------------------------------------------------
   #* LocoTest.tcl - Test Program for the Rail Driver Daemon.
   #* Created by Robert Heller on Sat May  5 10:29:48 2007
6  #* ------------------------------------------------------------------------
   #* Modification History: $Log$
   #* Modification History: Revision 1.3  2007/05/06 19:14:57
   heller
   #* Modification History: Lock down  for 2.1.8 release candidate 1
   #* Modification History:
11 #* Modification History: Revision 1.1  2007/05/06 13:45:41
   heller
   #* Modification History: Lock down  for 2.1.8 release candidate 1
   #* Modification History:
   #* Modification History: Revision 1.1  2002/07/28 14:03:50
   heller
   #* Modification History: Add it copyright notice headers
16 #* Modification History:
   #* ------------------------------------------------------------------------
   #* Contents:
   #* ------------------------------------------------------------------------
   #*
21 #*      Model RR System, Version 2
   #*      Copyright (C) 1994,1995,2002-2007  Robert Heller
   #*                     D/B/A Deepwoods Software
   #*                     51 Locke Hill Road
   #*                     Wendell, MA 01379-9728
26 #*
   #*      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., 675 Mass Ave, Cambridge, MA 02139, USA.
#*
#*
#*

# Global variables containing current settings.
global Direction Throttle IndepentBrake CurrentSpeed

# Initial speed is 0 (stopped)
set CurrentSpeed 0
# Initial direction is 0 (neutral)
set Direction 0
# Initial throttle setting is 0 (idle)
set Throttle 0
# Initial independent brake setting is 0 (released)
set IndepentBrake 0

# Constant: maximim allowed speed.
global MaximumSpeed
# Max simulated speed is 60 (mph)
set MaximumSpeed 60

# Alert button flag
global Alert
set Alert 0

# Tables that map lever positions to throttle and
# brake application values
global ThrottleValues IndepentBrakeValues

array set ThrottleValues {
   0 0   1 0   2 0   3 0   4 0   5 0   6 0   7 0
   8 0   9 0  10 0  11 0  12 0  13 0  14 0  15 0
  16 0  17 0  18 0  19 0  20 0  21 0  22 0  23 0
  24 0  25 0  26 0  27 0  28 0  29 0  30 0  31 0
  32 0  33 0  34 0  35 0  36 0  37 0  38 0  39 0
```

```
      40   0   41   0   42   0   43   0   44   0   45   0   46   0   47   0
 76   48   0   49   0   50   0   51   0   52   0   53   0   54   0   55   0
      56   0   57   0   58   0   59   0   60   0   61   0   62   0   63   0
      64   0   65   0   66   0   67   0   68   0   69   0   70   0   71   0
      72   0   73   0   74   0   75   0   76   0   77   0   78   0   79   0
      80   0   81   0   82   0   83   0   84   0   85   0   86   0   87   0
 81   88   0   89   0   90   0   91   0   92   0   93   0   94   0   95   0
      96   0   97   0   98   0   99   0  100   0  101   0  102   0  103   0
     104   0  105   0  106   0  107   0  108   0  109   0  110   0  111   0
     112   0  113   0  114   0  115   0  116   0  117   0  118   0  119   0
     120   0  121   0  122   0  123   0  124   0  125   0  126   0  127   0
 86  128   0  129   0  130   0  131   0  132   0  133   0  134   0  135   0
     136   0  137   0  138   0  139   0  140   0  141   0  142   0  143   0
     144   0  145   0  146   0  147   0  148   0  149   0  150   0  151   0
     152   0  153   0  154   0  155   0  156   0  157   0  158   1  159   1
     160   2  161   2  162   2  163   3  164   3  165   4  166   4  167   4
 91  168   5  169   5  170   6  171   6  172   6  173   7  174   7  175   8
     176   8  177   9  178   9  179   9  180  10  181  10  182  11  183  11
     184  11  185  12  186  12  187  13  188  13  189  13  190  14  191  14
     192  15  193  15  194  16  195  16  196  16  197  17  198  17  199  18
     200  18  201  18  202  19  203  19  204  20  205  20  206  20  207  21
 96  208  21  209  22  210  22  211  23  212  23  213  23  214  24  215  24
     216  25  217  25  218  25  219  26  220  26  221  27  222  27  223  27
     224  28  225  28  226  29  227  29  228  30  229  30  230  30  231  30
     232  30  233  30  234  30  235  30  236  30  237  30  238  30  239  30
     240  30  241  30  242  30  243  30  244  30  245  30  246  30  247  30
101  248  30  249  30  250  30  251  30  252  30  253  30  254  30
}
array set IndepentBrakeValues {
       0 30    1 30    2 30    3 30    4 30    5 30    6 30    7 30
       8 30    9 30   10 30   11 30   12 30   13 30   14 30   15 30
106   16 30   17 30   18 30   19 30   20 30   21 30   22 30   23 30
      24 30   25 30   26 30   27 30   28 30   29 30   30 30   31 30
      32 30   33 30   34 30   35 30   36 30   37 30   38 30   39 30
      40 30   41 30   42 29   43 29   44 29   45 29   46 29   47 29
      48 28   49 28   50 28   51 28   52 28   53 28   54 27   55 27
111   56 27   57 27   58 27   59 26   60 26   61 26   62 26   63 26
      64 26   65 25   66 25   67 25   68 25   69 25   70 25   71 24
      72 24   73 24   74 24   75 24   76 23   77 23   78 23   79 23
      80 23   81 23   82 22   83 22   84 22   85 22   86 22   87 22
      88 21   89 21   90 21   91 21   92 21   93 21   94 20   95 20
116   96 20   97 20   98 20   99 19  100 19  101 19  102 19  103 19
     104 19  105 18  106 18  107 18  108 18  109 18  110 18  111 17
     112 17  113 17  114 17  115 17  116 16  117 16  118 16  119 16
     120 16  121 16  122 15  123 15  124 15  125 15  126 15  127 15
```

```
     128 14 129 14 130 14 131 14 132 14 133 13 134 13 135 13
121  136 13 137 13 138 13 139 12 140 12 141 12 142 12 143 12
     144 12 145 11 146 11 147 11 148 11 149 11 150 11 151 10
     152 10 153 10 154 10 155 10 156  9 157  9 158  9 159  9
     160  9 161  9 162  8 163  8 164  8 165  8 166  8 167  8
     168  7 169  7 170  7 171  7 172  7 173  6 174  6 175  6
126  176  6 177  6 178  6 179  5 180  5 181  5 182  5 183  5
     184  5 185  4 186  4 187  4 188  4 189  4 190  3 191  3
     192  3 193  3 194  3 195  3 196  2 197  2 198  2 199  2
     200  2 201  2 202  1 203  1 204  1 205  1 206  1 207  1
     208  0 209  0 210  0 211  0 212  0 213  0 214  0 215  0
131  216  0 217  0 218  0 219  0 220  0 221  0 222  0 223  0
     224  0 225  0 226  0 227  0 228  0 229  0 230  0 231  0
     232  0 233  0 234  0 235  0 236  0 237  0 238  0 239  0
     240  0 241  0 242  0 243  0 244  0 245  0 246  0 247  0
     248  0 249  0 250  0 251  0 252  0 253  0 254  0
136 }

    # Contains the file  channel connected to the daemon process
    global Socket

141 # Connect to server
    set Socket [socket localhost 41000]
    # Line buffering
    fconfigure $Socket -buffering line

146 # File event to asyncronously handle messages from the server
    fileevent $Socket readable {
            # Get a line from the server.
            # If EOF, close the connection and exit.
            if {[gets $Socket line] < 0} {
151             close $Socket
                after 1000 exit
            } else {
              # Reverser lever moved?  Get its setting and save in Direction
              if {$CurrentSpeed == 0 &&
156               [regexp {REVERSER=([0-9]*)} "$line" -> reverser] > 0} {
                if {$reverser < 64} {
                  set Direction 1
                } elseif {$reverser > 220} {
                  set Direction -1
161             } elseif {$reverser < 128 && $reverser > 100} {
                  set Direction 0
                  set Throttle 0
                }
```

```
            }
166         # Throttle lever moved?  Fetch its value.
            if {[regexp {THROTTLE=([0-9]*)} "$line" -> throttle] > 0} {
              if {$Direction != 0} {
                set Throttle $ThrottleValues($throttle)
              }
171         }
            # Indepent Brake lever moved?  Fetch its value.
            if {[regexp {INDEPENDBRK=([0-9]*)} "$line" -> ibrake] > 0} {
              set IndepentBrake $IndepentBrakeValues($ibrake)
            }
176         # Check for Alert button (one of the bits in DIGITAL5).
            #puts stderr "*** line = $line"
            if {[regexp {DIGITAL5=\((.*)\)} "$line" -> d5] > 0} {
              # Alert button pressed?  Flag it.
              #puts stderr "*** d5 = $d5"
181           if {[string first Alert "$d5"] >= 0} {
                set Alert 1
              }
            }
          }
186
    }

    # This proc is run every 1/2 second and simulates a locomotive.
    proc Update {} {
191       # Globals
          global Direction Throttle IndepentBrake CurrentSpeed
          global MaximumSpeed
          global Socket
          global Alert
196
          # Was the Alert button pressed?  If so, reset the speedometer
          # display and close the connection to the server
          if {$Alert} {
            puts $Socket "LED 000"
201         flush $Socket
            puts $Socket "Exit"
            flush $Socket
            return
          }
206       # If the reverser is not in neutral, apply the throttle's
          # acceleration.
          if {$Direction != 0} {
            incr CurrentSpeed $Throttle
```

```
        }
211     incr CurrentSpeed -1;#  Entropy (rolling friction)
        incr CurrentSpeed -$IndepentBrake;# Brake Application
        # Apply the governer (speed limit)
        if {$CurrentSpeed > $MaximumSpeed} {
          set CurrentSpeed $MaximumSpeed
216     }
        # Avoid a negative speed
        if {$CurrentSpeed < 0} {set CurrentSpeed 0}
        # Format the speed for  display.
        set digits [format {%2d} $CurrentSpeed]
221     regsub -all { } "$digits" {_} digits
        # Apply direction.
        if {$Direction < 0} {
          set digits "-$digits"
        } else {
226       set digits "_$digits"
        }
        # Update the speedometer.
        puts $Socket "LED_$digits"
        flush $Socket
231     # Fetch new values.
        puts $Socket "POLLVALUES_REVERSER_THROTTLE_INDEPENDBRK_DIGITAL5"
        flush $Socket

        # Sleep for 1/2 second and check again.
236     after 500 Update
}


# Initial poll
241 puts $Socket "POLLVALUES_REVERSER_THROTTLE_INDEPENDBRK_DIGITAL5"
flush $Socket
# Sleep for 1/2 second and check.
after 500 Update

246 # Enter event loop.
vwait forever
```

The Tcl script shown in Listing 3.1 is a simple Tcl program that connects to the Rail Driver daemon and simulates a locomotive, by using the reverser, throttle, and independent brake levers to set the direction, accelerate, and slow down a virtual locomotive and display its speed on the LED display on the Rail Driver console.

A connection to the Rail Driver daemon is opened on line 136 and the resulting

file channel is set to line buffering on line 138. Lines 141–179 contain a file event handler for messages coming from the daemon. This handler parses the results and updates the global state variables based on changes in the reverser, throttle, and independent brake levers. It also checks for the Alert button and sets a flag if this button is pressed. If it looses the connection to the server, it closes the connection and arranging to exit the program.

Lines 181–228 contain a procedure that is run every 1/2 second. This procedure applies updates to the virtual locomotive's state[1], taking into account the current settings of the reverser, throttle, and independent brake levers. It then sends the current state (speed) to the LED display on the Rail Driver console. It also checks the Alert flag and arranges to disconnect from the daemon when the Alert button is pressed.

[1]Only its speed, contained in the global variable `CurrentSpeed`. To control an actual locomotive, this is where there would be code (eg using the XPressNet class library) to set the locomotive's speed.

Part II

Azatrax (From C++ and Tcl)

Chapter 4

Using the Azatrax device Interfaces.

4.1 aztrax::Azatrax Class

Listing 4.1: azatrax::Azatrax Class (simplified public interface)

```
class aztrax::Azatrax {
public:
        ~Azatrax();
        static int NumberOfOpenDevices();
        static char ** AllConnectedDevices();
        static Azatrax *OpenDevice(const char *serialnumber,
                                   unsigned short int idProduct=0,
                                   char **outmessage=NULL);
        friend class aztrax::MRD;
        friend class aztrax::SL2;
        friend class aztrax::SR4;
        enum {
                idAzatraxVendor,
                idMRDProduct,
                idSL2Product,
                idSR4Product
        };
        ErrorCode RestoreLEDFunction() const;
        ErrorCode Identify_1() const;
        ErrorCode GetStateData();
        uint8_t PacketCount() const;
        const char *SerialNumber() const;
        const char *MyProduct() const;
        unsigned short int MyProductId() const;
        static unsigned short int ProductIdCode(const char *productName);
```

```
};
```

The Azatrax Interface[4] has both a C++ and a Tcl API and this chapter will cover both. Both APIs use the same underlying code[1]. The API is based on the C++ classes that encapsulates the various Azatrax devices, the MRD2-S or MRD2-U (MRD class), the SR4 (the MRD class), or the SL2 (the SL2 class). Each Azatrax device has a factory set serial number.

The Azatrax class is the base class for all of the Azatrax device classes. Its constructor should not be called directly. Instead the `OpenDevice` static member should be called. This member function will return a device specific instance (at this time, one of aztrax::MRD, aztrax::SR4, or aztrax::SL2), depending on the type of device opened. The device will be opened based on its serial number and optionally by a specific product type. The result of `OpenDevice` function should be cast to a specific device class. It will never really be a base Azatrax class instance, since the base class has little functional usefulness. The device specific classes implement both the common functionallity of the base class and all of the device specific functionallity.

4.2 aztrax::MRD Class

Listing 4.2: azatrax::MRD Class (simplified public interface)

```
#define ErrorCode int
class azatrax::MRD public azatrax::Azatrax{
public:
    enum OperatingMode_Type { NonTurnoutSeparate,
                              NonTurnoutDirectionSensing,
                              TurnoutSolenoid,
                              TurnoutMotor };
    ~MRD();
    ErrorCode SetChan1() const;
    ErrorCode SetChan2() const;
    ErrorCode ClearExternallyChanged() const;
    ErrorCode DisableExternal() const;
    ErrorCode EnableExternal() const;
    ErrorCode RestoreLEDFunction() const;
    ErrorCode Identify_2() const;
    ErrorCode Identify_1_2() const;
    ErrorCode ResetStopwatch() const;
    bool Sense_1() const;
    bool Sense_2() const;
```

[1] In fact the same shared library.

```
        bool Latch_1() const;
        bool Latch_2() const;
        bool HasRelays() const;
        bool ResetStatus() const;
24      bool StopwatchTicking() const;
        bool ExternallyChanged() const;
        bool AllowingExternalChanges() const;
        OperatingMode_Type OperatingMode() const;
        void Stopwatch(uint8_t &fract, uint8_t &seconds, uint8_t &minutes,
29                     uint8_t &hours) const; };
```

The MRD2-S has two relays and the MRD2-U has no relays. Both devices have two sensors and a stopwatch. The class instance is connected to a device using its unique serial number, using the aztrax::Azatrax static method, `OpenDevice`, see Section 4.1, and Listing 4.1. This class exposes a collection of methods to send command bytes to the device, a method to retrieve the device's state data and a collection of methods to access the device's state data, once it has been retrieved.

Part III

Tcl Script Programming

Chapter 5

Using the CMR/I (Bruce Chubb) Interface.

5.1 CMri Class

Listing 5.1: CMri SNIT type (Class), simplified public interface

```
snit::type CMri {
  option -baud -readonly yes -default 9600 \
               -type {snit::enum -values {9600 19200 28800 57600 115200}}
  option -maxtries -readonly yes -default 10000 \
               -type {snit::integer -min 1000 -max 100000}
  constructor {port args} { }
  destructor { }
  method Inputs {ni {ua 0}} { }
  method Outputs {ports {ua 0}} { }
  method InitBoard {CT ni no ns ua card dl} { }
}
```

The CMR/I Interface[4] has a Tcl API encoded as a Tcl SNIT type (class) that encapsulates a Bruce Chubb CMR/I[1, 3] serial "bus", containing one or more interface nodes, which are instances of SMINI (Super Mini Node), USIC (Classic Universal Serial Interface Card), or SUSIC (Super Classic Universal Serial Interface Card). Each card has a unique 6 bit address and contains a number of input and output ports[1]. The SNIT type (class) connects to a serial port which has an RS232 to RS485 conversion card that in turn connects to the RS485-based serial bus. This SNIT type (class) exposes three methods to access cards on the bus,

[1]The SMINI is a self contained card with 3 input ports (24 bits) and 6 output ports (48 bits).

an initialization method to initialize the board, an input method to read the input ports, and an output method to write to the output ports. A simplified listing of this SNIT type (class) is shown in Listing 5.1.

The constructor takes the name of the port device file, with options for the BAUD rate and the maximum number of retries. The constructor opens the serial port and conditions it for unbuffered binary I/O at the specified BAUD rate. The destructor restores the port and closes it. The Inputs() method reads the contents of the input ports of a specified node and the Outputs() method writes to the output ports of a specified node. The InitBoard() method initializes a specified node. The InitBoard takes either a packed card type list (for USIC or SUSIC nodes) or a yellow bi-color LED map (for SMINI nodes), the number of input[2] and output[3] ports, the number of yellow bi-color LED signals[4], the card address, delay value to use[5], and the error message pointer.

5.2 Basic usage

Listing 5.2: Using the CMR/I from Tcl

```
#
package require Cmri 2.0.0;#                Load the CMR/I snit type.

# The board address of the SMINI card.
global UA
set UA 0


# Connect to the bus on COM3: (/dev/ttyS2), at 9600 BAUD, with
# a retry count of 10000, capturing error messages.
if {[catch {cmri::CMri CMriBus /dev/ttys0 -baud 9600 -maxtries 10000} errorMessa
  # Handle error.
  puts -nonewline stderr "Could not connect to CMR/I bus on /dev/ttyS2: "
  puts stderr "$errorMessage"
  rename CMriBus {}
  exit 99
}

# Initialize SMINI board
if {[catch {CMriBus InitBoard {0 0 0 0 0 0} 3 6 0 $UA SMINI 0} result]} {
```

[2]Must be 3 for SMINI cards.
[3]Must be 6 for SMINI cards.
[4]Only used for SMINI cards.
[5]Only used for for older USIC cards.

```
    set errorMessage [lindex $result 1]
    # Handle error.
23  puts -nonewline stderr "Could not initialize SMINI card at UA "
    puts stderr "$UA: $errorMessage"
    rename CMriBus {}
    exit 99
}
28
# Initialize port buffer
#       Turn all bits of port 0 on
#       Turn all of the even bits of port 1 on
#       Turn all of the odd bits of port 2 on
33  #       Turn the lower nibble of port 3 on
#       Turn the upper nibble of port 4 on
#       Leave port 5 off
  set Outputs {0xFF 0xAA 0x55 0x0F 0xF0 0}
# Write to SMINI ports
38 if {[catch {CMriBus Outputs $Outputs $UA} result]} {
  # Handle error.
  puts -nonewline stderr "Could not write to the output ports of "
  puts stderr "SMINI card at UA $UA: $result"
  rename CMriBus {}
43  exit 99
}

# Read SMINI ports
if {[catch {CMriBus Inputs 3 $UA} result]} {
48  puts -nonewline stderr "Could not read from the input ports of "
  puts stderr "SMINI card at UA $UA: $result"
  rename CMriBus {}
  exit 99
}
53
set Inputs $result

foreach p {0 1 2} v $Inputs {
  puts "Port $p = $v"
58 }

rename CMriBus {}
exit
```

The program shown in Listing 5.2 opens a connection to a CMR/I bus with a SMINI card (Line 10 of Listing 5.2). It then initialize the SMINI card (Line 20 of Listing 5.2), write to the output ports of the SMINI card (Line 38 of List-

ing 5.2), and then then read from the input ports of the SMINI card (Line 48 of Listing 5.2). Finally, the serial port is closed when the instance is deleted (Line 62 of Listing 5.2) and the program exits. While this is valid program, it is not partitularly useful in a Model Railroad control context. More typically, the program would have a loop around the I/O functions, repeatedly reading the inputs, which would typically contain information about the state of CTC panel controls, track switches (turnouts), and block occupancy detectors, and code to set the outputs, which would control turnouts (via switch machines), track side signals, and CTC panel lamps[6]. A more realistic program is shown in skeleton form in Listing 5.3.

5.3 Advanced usage

Listing 5.3: Using the CMR/I from Tcl, more realistic version

```
#
package require Cmri 2.0.0;#              Load the CMR/I shared library.

# The board address of the SMINI card.
global UA
set UA 0


# Connect to the bus on COM3: (/dev/ttyS2), at 9600 BAUD, with
# a retry count of 10000, capturing error messages.
if {[catch {cmri::CMri CMriBus /dev/ttys0 -baud 9600 -maxtries 10000} errorMessa
  # Handle error.
  puts -nonewline stderr "Could not connect to CMR/I bus on /dev/ttyS2: "
  puts stderr "$errorMessage"
  rename CMriBus {}
  exit 99
}

# Initialize SMINI board
if {[catch {CMriBus InitBoard {0 0 0 0 0 0} 3 6 0 $UA SMINI 0} result]} {
  set errorMessage [lindex $result 1]
  # Handle error.
  puts -nonewline stderr "Could not initialize SMINI card at UA "
  puts stderr "$UA: $errorMessage"
  rename CMriBus {}
  exit 99
}
```

[6] See Chapter 11 for information about programming CTC Panels on a computer screen using Tcl/Tk.

```
29  while {1} {
      # Read SMINI ports
      if {[catch {CMriBus Inputs 3 $UA} errorMessage]} {
        puts -nonewline stderr "Could not read from the input ports of "
        puts stderr "SMINI card at UA $UA: $errorMessage"
34    } else {
        set Inputs $result
        # Process Input ports, generating values for the output ports
        # (Process code goes here)
        # Write to SMINI ports
39      if {[catch {CMriBus Outputs $Outputs $UA} errorMessage]} {
          # Handle error.
          puts -nonewline stderr "Could not write to the output ports of "
          puts stderr "SMINI card at UA $UA: $errorMessage"
        }
44    }
      after 1000;# Sleep for one second
    }
    rename CMriBus {}
    exit
```

5.4 Translations from The C/MRI User's Manual V3.0 Examples

I have translated selected example programs from The C/MRI User's Manual V3.0[2]. The source code files are included in the documentation directory.

5.4.1 From Chapter 9: SMINI Application Examples

Listing 5.4: Figure 9-5, Tcl Translation

```
    # $Id: fig9-5.tcl 1465 2013-03-17 15:24:03Z heller $
2
    # Load MRR System packages
    # Add MRR System package Paths
    lappend auto_path /usr/local/chare/MRRSystem;# Tcl packages
    package require Cmri 2.0.0;#           Load the CMR/I package
7
    #  REM**DEFINE CONSTANTS FOR PACKING AND UNPACKING I/O BYTES
    #      B0 = 1: B1 = 2: B2 = 4: B3 = 8: B4 = 16: B5 = 32: B6 = 64: B7 = 128
    #      W1 = 1: W2 = 3: W3 = 7: W4 = 15: W5 = 31: W6 = 63: W7 = 127
```

```
# Bit shift constants
#
set B0 0
set B1 1
set B2 2
set B3 3
set B4 4
set B5 5
set B6 6
set B7 7

# Bit mask constants
#
set W1 0x01
set W2 0x03
set W3 0x07
set W4 0x0f
set W5 0x1f
set W6 0x3f
set W7 0x7f


#  REM**DEFINE BLOCK OCCUPATION CONSTANTS
#      CLR = 0          'Clear
#      OCC = 1          'Occupied

set CLR 0
set OCC 1

#  REM**DEFINE SIGNAL ASPECTS
#      DRK = 0          'Dark   00
#      GRN = 1          'Green  01
#      RED = 2          'Red    10

set DRK 0                   # Dark   00
set GRN 1                   # Green  01
set RED 2                   # Red    10

set UA 0;#     Address of our SMINI card

#  PRINT "SIGNALING LOOP TRACK USING SMINI WITH 2-ASPECT COLOR LIGHT SIGNALS"

puts "SIGNALING LOOP TRACK USING SMINI WITH 2-ASPECT COLOR LIGHT SIGNALS"

#  REM**INITIALIZE SMINI**
```

```
#       UA = 0              'USIC NODE ADDRESS
#       COMPORT = 2         'PC SERIAL COMMUNICATIONS PORT = 1, 2, 3 OR 4
#       BAUD100 = 96        'BAUD RATE OF 9600 DIVIDED BY 100
#       DL = 0              'USIC TRANSMISSION DELAY
#       NDP$ = "M"          'NODE DEFINITION PARAMETER
#       NS = 0              'NUMBER OF 2-LEAD SEARCHLIGHT SIGNALS
#       NI = 3                  'NUMBER OF INPUT PORTS
#       NO = 6                  'NUMBER OF OUTPUT PORTS
#       MAXTRIES = 10000  'MAXIMUM READ TRIES BEFORE ABORT INPUTS
#       GOSUB INIT     'INVOKE INITIALIZATION SUBROUTINE

#************************************
#* Initialize bus                   *
#************************************
# Connect to the bus on COM2: (/dev/ttyS1), at 9600 BAUD, with
# a retry count of 10000, capturing error messages.
if {[catch {cmri::CMri bus /dev/ttyS1 -baud 9600 -retries 10000} result]} {
        set errorMessage $result
        # Handle error.
        puts -nonewline stderr "Could not connect to CMR/I bus on /dev/ttyS1: "
        puts stderr "$errorMessage"
        rename bus {}
        exit 99
}
#************************************
#* Initialize board                 *
#************************************
if {[catch {bus InitBoard {0 0 0 0 0 0} 3 6 0 $UA SMINI 0} result]} {
        set errorMessage $result
        # Handle error.
        puts -nonewline stderr "Could not initialize SMINI card at UA "
        puts stderr "$UA: $errorMessage"
        rename bus {}
        exit 99
}

#       PRINT "NODE INITIALIZATION IS COMPLETE - CHECK LED BLINK RATE"
#       PRINT "    AND PRESS ANY KEY TO CONTINUE"
#       SLEEP

puts "NODE INITIALIZATION IS COMPLETE - CHECK LED BLINK RATE"
puts "   AND PRESS ANY KEY TO CONTINUE"
gets stdin

#BRTL:     '*******BEGIN REAL TIME LOOP*******
```

```
102
    while {true} {

    #  REM**READ INPUT BYTES FROM SMINI's 3 INPUT PORTS
    #     GOSUB INPUTS     'Input bytes are stored as IB(1), IB(2), IB(3)
107
          if {[catch {bus Inputs 3 $UA} result]} {
                set errorMessage $result
                puts -nonewline stderr "Could not read from the input ports of "
                puts stderr "SMINI card at UA $UA: $errorMessage"
112             break
          }
          set Inputs $result
    #
    #  REM**UNPACK INPUTS
117 #     BK(1) = IB(1) \ B0 AND W1   'CARD 2 PORT A
    #     BK(2) = IB(1) \ B1 AND W1
    #     BK(3) = IB(1) \ B2 AND W1
    #     BK(4) = IB(1) \ B3 AND W1
    #     BK(5) = IB(1) \ B4 AND W1
122 #     BK(6) = IB(1) \ B5 AND W1

          set BK(0) [expr {[lindex $Inputs 0] << $B0 & $W1}];# CARD 2 PORT A
          set BK(1) [expr {[lindex $Inputs 0] << $B1 & $W1}]
          set BK(2) [expr {[lindex $Inputs 0] << $B2 & $W1}]
127       set BK(3) [expr {[lindex $Inputs 0] << $B3 & $W1}]
          set BK(4) [expr {[lindex $Inputs 0] << $B4 & $W1}]
          set BK(5) [expr {[lindex $Inputs 0] << $B5 & $W1}]
    #
    #  REM**INITIALIZE ALL SIGNALS TO GREEN
132 #     FOR I = 1 TO 6: SE(I) = GRN: SW(I) = GRN: NEXT I

          for {set i 0} {$i < 6} {incr i} {set SE($i) $GRN;set SW($i) $GRN}

    #
137 # REM**CHECK IF BLOCK OCCUPIED THEN SET SIGNALS LEADING INTO BLOCK RED
    #     IF BK(1) = OCC THEN SE(6) = RED: SW(2) = RED
    #     IF BK(2) = OCC THEN SE(1) = RED: SW(3) = RED
    #     IF BK(3) = OCC THEN SE(2) = RED: SW(4) = RED
    #     IF BK(4) = OCC THEN SE(3) = RED: SW(5) = RED
142 #     IF BK(5) = OCC THEN SE(4) = RED: SW(6) = RED
    #     IF BK(6) = OCC THEN SE(5) = RED: SW(1) = RED

          if {$BK(0) == $OCC} {set SE(5) $RED;set SW(1) $RED}
          if {$BK(1) == $OCC} {set SE(0) $RED;set SW(2) $RED}
```

```
147         if {$BK(2) == $OCC} {set SE(1) $RED;set SW(3) $RED}
            if {$BK(3) == $OCC} {set SE(2) $RED;set SW(4) $RED}
            if {$BK(4) == $OCC} {set SE(3) $RED;set SW(5) $RED}
            if {$BK(5) == $OCC} {set SE(4) $RED;set SW(0) $RED}

152 #
    # REM**IMPLEMENT APPROACH LIGHTING BY SETTING SIGNALS TO DARK...
    # REM**       ...IF BLOCK APPROACHING SIGNAL IS CLEAR
    #     FOR I = 1 TO 6
    #        IF BK(I) = CLR THEN SE(I) = DRK: SW(I) = DRK
157 #     NEXT I

          for {set i 0} {$i < 6} {incr i} {
                  if {$BK(i) == $CLR} {set SE(i) $DRK; set SW(i) $DRK}
          }
162
    #
    #  REM**PACK OUTPUT BYTES
    #     OB(1) = SE(1)                          'SMINI CARD 0 PORT A
    #     OB(1) = SW(1) * B2 OR OB(1)
167 #     OB(1) = SE(2) * B4 OR OB(1)
    #     OB(1) = SW(2) * B6 OR OB(1)

          # SMINI CARD 0 PORT A
          set Outputs [expr {$SE(0) |                \
172                          $SW(0) << $B2 |         \
                             $SE(1) << $B4 |         \
                             $SW(1) << B6}]

    #
177 #     OB(2) = SE(3)                          'SMINI CARD 0 PORT B
    #     OB(2) = SW(3) * B2 OR OB(2)
    #     OB(2) = SE(4) * B4 OR OB(2)
    #     OB(2) = SW(4) * B6 OR OB(2)

182       # SMINI CARD 0 PORT B
          lappend Outputs [expr {$SE(2) |            \
                                 $SW(2) << $B2 |     \
                                 $SE(3) << $B4 |     \
                                 $SW(3) << $B6}]
187
    #
    #     OB(3) = SE(5)                          'SMINI CARD 0 PORT C
    #     OB(3) = SW(5) * B2 OR OB(3)
    #     OB(3) = SE(6) * B4 OR OB(3)
```

```
192 #       OB(3) = SW(6) * B6 OR OB(3)

              # SMINI CARD 0 PORT C
              lappend Outputs [expr {$SE(4) |          \
                                     $SW(4) << $B2 |   \
197                                  $SE(5) << $B4 |   \
                                     $SW(5) << $B6}]

              lappend Outputs 0 0 0;  # SMINI CARD 1 PORTs A, B, C
    #
202 #   REM**WRITE OUTPUT BYTES TO SMINI's 6 OUTPUT PORTS
    #       GOSUB OUTPUTS

              if {[catch {bus Outputs $Outputs $UA} result]} {
                    set errorMessage $result
207                 # Handle error.
                    puts -nonewline stderr "Could not write to the output ports of "
                    puts stderr "SMINI card at UA $UA: $errorMessage"
                    break
              }
212 #
    #   REM**RETURN TO BEGINNING OF REAL-TIME LOOP
    #       GOTO BRTL

    }
```

5.4.2 From Chapter 12: SUSIC/USIC Application Examples

Listing 5.5: Figure 12-13, Tcl Translation

```
  2 # Load MRR System packages
    # Add MRR System package Paths
    lappend auto_path /usr/local/share/MRRSystem;# Tcl packages
    package require Cmri 2.0.0;#            Load the CMR/I package

  7 #   REM**SUSIC SINGLE-NODE SYSTEM USING 3-ASPECT COLOR LIGHT SIGNALS**
    #   REM**DEFINE VARIABLE TYPES AND ARRAY SIZES
    #       DEFINT A-Z
    #         DIM OB(60), IB(60), TB(80), CT(15)
    #
 12 #   REM**DEFINE CONSTANTS FOR PACKING AND UNPACKING I/O BYTES
    #       B0 = 1: B1 = 2: B2 = 4: B3 = 8: B4 = 16: B5 = 32: B6 = 64: B7 = 128
    #       W1 = 1: W2 = 3: W3 = 7: W4 = 15: W5 = 31: W6 = 63: W7 = 127

    # Bit shift constants
```

```
#
set B0 0
set B1 1
set B2 2
set B3 3
set B4 4
set B5 5
set B6 6
set B7 7

# Bit mask constants
#
set W1 0x01
set W2 0x03
set W3 0x07
set W4 0x0f
set W5 0x1f
set W6 0x3f
set W7 0x7f

#
#  REM**DEFINE BLOCK OCCUPATION CONSTANTS
#      CLR = 0          'Clear
#      OCC = 1          'Occupied

set CLR 0
set OCC 1

#
#     REM**DEFINE SIGNAL ASPECTS - Single Head
#      DRK = 0    'Dark     000
#      GRN = 1    'Green    001
#      YEL = 2    'Yellow   010
#      RED = 4    'Red      100

set DRK 0;# Dark     000
set GRN 1;# Green    001
set YEL 2;# Yellow   010
set RED 4;# Red      100

#
#  REM**DEFINE SIGNAL ASPECTS - Double Head
#      GRNRED = 17    'Green over red     10001
#      YELRED = 18    'Yellow over red    10010
#      REDYEL = 12    'Red over yellow    01100
```

```
#     REDRED = 20    'Red over red       10100

set GRNRED 0x11;# Green over red     10001
set YELRED 0x12;# Yellow over red    10010
set REDYEL 0x0c;# Red over yellow    01100
set REDRED 0x14;# Red over red       10100

#
#  REM**DEFINE SIGNAL ASPECTS - Triple Head
#     REDREDRED = 84    'Red over red over red      1010100
#     REDREDYEL = 52    'Red over red over yellow   0110100
#     REDYELRED = 76    'Red over yellow over red   1001100
#     YELREDRED = 82    'Yellow over red over red   1010010
#     GRNREDRED = 81    'Green over red over red    1010001

set REDREDRED 0x54;# Red over red over red      1010100
set REDREDYEL 0x34;# Red over red over yellow   0110100
set REDYELRED 0x4c;# Red over yellow over red   1001100
set YELREDRED 0x52;# Yellow over red over red   1010010
set GRNREDRED 0x51;# Green over red over red    1010001

#
#  REM**DEFINE TURNOUT POSITION
#  REM**Constants provided below assume SMC card connected between
#  REM**switchmotors and a single output pin on DOUT32 cards. See
#  REM**other examples for direct connections using 2 output pins
#     TUN = 0
#     TUR = 1

set TUN 0
set TUR 1

#
#  REM**DEFINE PUSHBUTTON AND TOGGLE POSITIONS
#     PBP = 1    'Pushbutton pressed
#     TGR = 1    'Toggle reverse position
#     TGN = 0    'Toggle normal position

set PBP 1;# Pushbutton pressed
set TGR 1;# Toggle reverse position
set TGN 0;# Toggle normal position

#
#  REM**DEFINE DIRECTION-OF-TRAFFIC RUNNING ON SINGLE TRACK
#     NDT = 0       'No direction-of-traffic defined on single track
```

```
#       EBD = 1         'EastBounD Direction
#       WBD = 2         'WestBounD Direction

set NDT 0;# No direction-of-traffic defined on single track
set EBD 1;# EastBounD Direction
set WBD 2;# WestBounD Direction

set UA 0     /* Address of our SUSIC card. */

#
#  REM**INITIALIZE DIRECTION-OF-TRAFFIC VARIABLES TO NO DIRECTION-OF-TRAFFIC
#       DOT1 = NDT: DOT2 = NDT: DOT3 = NDT
          set DOT1 $NDT; set DOT2 $NDT; set DOT3 $NDT

#
#  PRINT "SINGLE NODE RAILROAD EXAMPLE USING 3-ASPECT COLOR LIGHT SIGNALS"
#

          puts "SINGLE NODE RAILROAD EXAMPLE USING 3-ASPECT COLOR LIGHT SIGNALS"

#  REM**INITIALIZE SUSIC
#       UA = 0              'USIC NODE ADDRESS
#       COMPORT = 3         'PC SERIAL COMMUNICATIONS PORT = 1, 2, 3 OR 4
#       BAUD100 = 192       'BAUD RATE OF 19200 DIVIDED BY 100
#       DL = 0              'USIC TRANSMISSION DELAY
#       NDP$ = "X"          'NODE DEFINITION PARAMETER
#       NS = 2              'NUMBER OF CARD SETS OF 4 (OIIO OOXX)
#       CT(1) = 150         'CARD TYPE SET OF (OIIO)
#       CT(2) = 10          'CARD TYPE SET OF (OOXX)
#       NI = 8                  'NUMBER OF INPUT PORTS
#       NO = 16                 'NUMBER OF OUTPUT PORTS
#       MAXTRIES = 10000    'MAXIMUM READ TRIES BEFORE ABORT INPUTS
#       GOSUB INIT          'INVOKE INITIALIZATION SUBPROGRAM

#************************************
#* Initialize bus                   *
#************************************
# Connect to the bus on COM2: (/dev/ttyS1), at 19200 BAUD, with
# a retry count of 10000, capturing error messages.
if {[catch {cmri::CMri bus /dev/ttyS1 -baud 19200 -retries 10000} errorMessage]} {
          # Handle error.
          puts -nonewline stderr "Could not connect to CMR/I bus on /dev/ttyS1: "
          puts stderr "$errorMessage"
          rename bus {}
          exit 99
```

```
152 }
    #************************************
    #* Initialize board                 *
    #************************************
    set CT [list 0x96 0x0A]
157         # 10010110
            #  O I I O
            # 00001010
            #  X X O O
    if {[catch {bus InitBoard $CT 8 16 0 $UA SUSIC 0} result]} {
162         # Handle error.
            puts -nonewline stderr "Could not initialize SUSIC card at UA "
            puts stderr "$UA: $result"
            rename bus {}
            exit 99
167 }
    #
    #     PRINT "NODE INITIALIZATION IS COMPLETE - CHECK LED BLINK RATE"
    #     PRINT "   AND PRESS ANY KEY TO CONTINUE"
    #     SLEEP
172
    puts "NODE INITIALIZATION IS COMPLETE - CHECK LED BLINK RATE"
    puts "   AND PRESS ANY KEY TO CONTINUE"
    gets stdin

177 #
    #BRTL: '*******BEGIN REAL TIME LOOP*******

    while {true} {

182 #  REM**READ AND UNPACK INPUTS
    #     GOSUB INPUTS
            if {[catch {bus Inputs 8 $UA} result]} {
                    puts -nonewline stderr "Could not read from the input ports of "
                    puts stderr "SUSIC card at UA $UA: $result"
187                 break
            }
            set Inputs $result

    #     BK1 = IB(1) \ B0 AND W1       'CARD 1 PORT A
192 #     BK2 = IB(1) \ B1 AND W1
    #     BK3 = IB(1) \ B2 AND W1
    #     BK4 = IB(1) \ B3 AND W1
    #     BK5 = IB(1) \ B4 AND W1
    #     BK6 = IB(1) \ B5 AND W1
```

```
197 #       OS1 = IB(1) \ B6 AND W1
    #       BK7 = IB(1) \ B7 AND W1

              set tempByte [lindex $Inputs 0];#         CARD 1 PORT A
              set BK1 [expr {$tempByte >> $B0 & $W1}]
202           set BK2 [expr {$tempByte >> $B1 & $W1}]
              set BK3 [expr {$tempByte >> $B2 & $W1}]
              set BK4 [expr {$tempByte >> $B3 & $W1}]
              set BK5 [expr {$tempByte >> $B4 & $W1}]
              set BK6 [expr {$tempByte >> $B5 & $W1}]
207           set OS1 [expr {$tempByte >> $B6 & $W1}]
              set BK7 [expr {$tempByte >> $B7 & $W1}]

    #
    #       BK8 = IB(2) \ B0 AND W1         'CARD 1 PORT B
212 #       OS2 = IB(2) \ B1 AND W1
    #       BK9 = IB(2) \ B2 AND W1
    #       BK10 = IB(2) \ B3 AND W1
    #       OS3 = IB(2) \ B4 AND W1
    #       BK11 = IB(2) \ B5 AND W1
217 #       OS4 = IB(2) \ B6 AND W1
    #       BK13 = IB(2) \ B7 AND W1

              set tempByte [lindex $Inputs 1];#         CARD 1 PORT B
              set BK8  [expr {$tempByte >> $B0 & $W1}]
222           set OS2  [expr {$tempByte >> $B1 & $W1}]
              set BK9  [expr {$tempByte >> $B2 & $W1}]
              set BK10 [expr {$tempByte >> $B3 & $W1}]
              set OS3  [expr {$tempByte >> $B4 & $W1}]
              set BK11 [expr {$tempByte >> $B5 & $W1}]
227           set OS4  [expr {$tempByte >> $B6 & $W1}]
              set BK13 [expr {$tempByte >> $B7 & $W1}]

    #
    #       BK14 = IB(3) \ B0 AND W1         'CARD 1 PORT C
232 #       OS5 = IB(3) \ B1 AND W1
    #       BK15 = IB(3) \ B2 AND W1
    #       BK20 = IB(3) \ B3 AND W1
    #       BK16 = IB(3) \ B4 AND W1
    #       BK17 = IB(3) \ B5 AND W1
237 #       OS6 = IB(3) \ B6 AND W1
    #       BK18 = IB(3) \ B7 AND W1

              set tempByte [lindex $Inputs 2];#         CARD 1 PORT C
              set BK14 [expr {$tempByte >> $B0 & $W1}]
```

```
242           set OS5  [expr {$tempByte >> $B1 & $W1}]
              set BK15 [expr {$tempByte >> $B2 & $W1}]
              set BK20 [expr {$tempByte >> $B3 & $W1}]
              set BK16 [expr {$tempByte >> $B4 & $W1}]
              set BK17 [expr {$tempByte >> $B5 & $W1}]
247           set OS6  [expr {$tempByte >> $B6 & $W1}]
              set BK18 [expr {$tempByte >> $B7 & $W1}]

      #
      #       TF12 = IB(4) \ B0 AND W1        'CARD 1 PORT D
252   #       PB1 = IB(4) \ B1 AND W1
      #       PB2 = IB(4) \ B2 AND W1
      #       PB3 = IB(4) \ B3 AND W1
      #       PB4 = IB(4) \ B4 AND W1
      #       PB5 = IB(4) \ B5 AND W1
257   #       PB6 = IB(4) \ B6 AND W1
      #       TG6 = IB(4) \ B7 AND W1

              set tempByte [lindex $Inputs 3];#          CARD 1 PORT D
              set TF12 [expr {$tempByte >> $B0 & $W1}]
262           set PB1  [expr {$tempByte >> $B1 & $W1}]
              set PB2  [expr {$tempByte >> $B2 & $W1}]
              set PB3  [expr {$tempByte >> $B3 & $W1}]
              set PB4  [expr {$tempByte >> $B4 & $W1}]
              set PB5  [expr {$tempByte >> $B5 & $W1}]
267           set PB6  [expr {$tempByte >> $B6 & $W1}]
              set TG6  [expr {$tempByte >> $B7 & $W1}]

      #
      #       TG7 = IB(5) \ B0 AND W1         'CARD 2 PORT A
272   #       TG8 = IB(5) \ B1 AND W1
      #       TG9 = IB(5) \ B2 AND W1
      #       TG10 = IB(5) \ B3 AND W1
      #       TG11 = IB(5) \ B4 AND W1

277           set tempByte [lindex $Inputs 4];#          CARD 2 PORT A
              set TG7  [expr {$tempByte >> $B0 & $W1}]
              set TG8  [expr {$tempByte >> $B1 & $W1}]
              set TG9  [expr {$tempByte >> $B2 & $W1}]
              set TG10 [expr {$tempByte >> $B3 & $W1}]
282           set TG11 [expr {$tempByte >> $B4 & $W1}]

      #
      #   REM**ALIGN PUSHBUTTON CONTROLLED TURNOUTS IN BUTTE IF OS1 IS CLEAR
      #       IF OS1 = OCC THEN GOTO TOGGLES
```

```
#       IF PB1 = PBP THEN SM4 = TUN: SM5 = TUN: TK = 1: GOTO TOGGLES
#       IF PB2 = PBP THEN SM4 = TUR: SM5 = TUN: TK = 2: GOTO TOGGLES
#       IF PB3 = PBP THEN SM5 = TUR: SM3 = TUR: TK = 3: GOTO TOGGLES
#       IF PB4 = PBP THEN SM5 = TUR: SM3 = TUN: SM2 = TUN: TK = 4: GOTO TOGGLES
#       IF PB5 = PBP THEN
#         SM5 = TUR: SM3 = TUN: SM2 = TUR: SM1 = TUR: TK = 5: GOTO TOGGLES
#       END IF
#       IF PB6 = PBP THEN SM5 = TUR: SM3 = TUN: SM2 = TUR: SM1 = TUN: TK = 6

          if {$OS1 != $OCC} {
            if {$PB1 == $PBP} {
              set SM4 $TUN; set SM5 $TUN; set TK 1
            } elseif {$PB2 == $PBP} {
              set SM4 $TUR; set SM5 $TUN; set TK 2
            } elseif {$PB3 == $PBP} {
              set SM5 $TUR; set SM3 $TUR; set TK 3
            } elseif {$PB4 == $PBP} {
              set SM5 $TUR; set SM3 $TUN; set SM2 $TUN; set TK 4
            } elseif {$PB5 == $PBP} {
              set SM5 $TUR; set SM3 $TUN; set SM2 $TUR; set SM1 $TUR; set TK 5
            } elseif {$PB6 == $PBP} {
              set SM5 $TUR; set SM3 $TUN; set SM2 $TUR; set SM1 $TUN; set TK 6
            }
          }

#
#  REM**ALIGN TOGGLE CONTROLLED TURNOUTS IF OS SECTIONS ARE CLEAR
#TOGGLES:
#       IF OS2 = CLR THEN SM6 = TG6
#       IF OS3 = CLR THEN SM7 = TG7
#       IF OS4 = CLR THEN SM8 = TG8: SM9 = TG9
#       IF OS5 = CLR THEN SM10 = TG10
#       IF OS6 = CLR THEN SM11 = TG11

          if {$OS2 == $CLR} {set SM6 $TG6}
          if {$OS3 == $CLR} {set SM7 $TG7}
          if {$OS4 == $CLR} {set SM8 $TG8; set SM9 $TG9}
          if {$OS5 == $CLR} {set SM10 $TG10}
          if {$OS6 == $CLR} {set SM11 $TG11}

#
#  REM**WHEN AN OS SECTION BECOMES OCCUPIED AND OPPOSING DIRECTION-
#  REM**OF-TRAFFIC IS NOT LOCKED-IN THEN SET DIRECTION-OF-TRAFFIC
#  REM**FOR SECTION OF SINGLE TRACK TO DIRECTION OF MOVEMENT
```

```
#       IF OS1 = OCC AND DOT1 <> WBD THEN DOT1 = EBD
#       IF OS2 = OCC AND DOT1 <> EBD THEN DOT1 = WBD
#       IF OS3 = OCC AND DOT2 <> WBD THEN DOT2 = EBD
#       IF OS4 = OCC AND DOT2 <> EBD THEN DOT2 = WBD
#       IF OS5 = OCC AND DOT3 <> WBD THEN DOT3 = EBD
#       IF OS6 = OCC AND DOT3 <> EBD THEN DOT3 = WBD

          if {$OS1 == $OCC && $DOT1 != $WBD} {set DOT1 $EBD}
          if {$OS2 == $OCC && $DOT1 != $EBD} {set DOT1 $WBD}
          if {$OS3 == $OCC && $DOT2 != $WBD} {set DOT2 $EBD}
          if {$OS4 == $OCC && $DOT2 != $EBD} {set DOT2 $WBD}
          if {$OS5 == $OCC && $DOT3 != $WBD} {set DOT3 $EBD}
          if {$OS6 == $OCC && $DOT3 != $EBD} {set DOT3 $WBD}

#  REM**RETAIN DIRECTIONT-OF-TRAFFIC WHILE SINGLE TRACK IS OCCUPIED
#DT1: IF OS1 = OCC THEN GOTO DT2
#       IF BK7 = OCC THEN GOTO DT2
#       IF BK8 = OCC THEN GOTO DT2
#       IF OS2 = OCC THEN GOTO DT2
#       DOT1 = NDT  'All trackage clear so set DOT1 to no direction-of-traffic

          if {$OS1 != $OCC &&
              $BK7 != $OCC &&
              $BK8 != $OCC &&
              $OS2 != $OCC} {
            set DOT1 $NDT; #All trackage clear so set DOT1 to no direction-of-traf
          }

#
#DT2: IF OS3 = OCC THEN GOTO DT3
#       IF BK11 = OCC THEN GOTO DT3
#       IF OS4 = OCC THEN GOTO DT3
#       DOT2 = NDT  'All trackage clear so set DOT2 to no direction-of-traffic

          if {$OS3 != $OCC &&
              $BK11 != $OCC &&
              $OS4 != $OCC} {
            set DOT2 $NDT; #All trackage clear so set DOT2 to no direction-of-traf
          }

#
#DT3: IF OS5 = OCC THEN GOTO DOTCMP
#       IF BK15 = OCC THEN GOTO DOTCMP
#       IF BK16 = OCC THEN GOTO DOTCMP
#       IF BK17 = OCC THEN GOTO DOTCMP
```

```
#       IF OS6 = OCC THEN GOTO DOTCMP
#       DOT1 = NDT  'All trackage clear so set DOT3 to no direction-of-traffic

          if {$OS5 != $OCC &&
              $BK15 != $OCC &&
              $BK16 != $OCC &&
              $BK17 != $OCC &&
              $OS6 != $OCC} {
            set DOT3 $NDT; #All trackage clear so set DOT3 to no direction-of-traffic
          }

#DOTCMP:
#
#   REM*****************************
#   REM*****************************
#   REM**CALCULATE EASTBOUND SIGNALS**
#   REM*****************************
#   REM*****************************
#
#   REM**Calculate SIG20RABC
#SIG20R:
#       SIG20RABC = REDREDRED
#       IF OS6 = OCC THEN GOTO SIG18R
#       IF SM11 = TUN AND TF12 = TGR THEN SIG20RABC = REDREDYEL: GOTO SIG18R
#       IF BK18 = OCC THEN GOTO SIG18R
#       IF SM11 = TUN THEN SIG20RABC = YELREDRED ELSE SIG20RABC = REDYELRED

          set SIG20RABC $REDREDRED
          if {$OS6 != $OCC} {
            if {$SM11 == $TUN && $TF12 == TGR} {
              set SIG20RABC $REDREDYEL
            } else {
              if {$BK18 != $OCC} {
                if (SM11 == TUN) {
                  set SIG20RABC $YELREDRED
                } else {
                  set SIG20RABC $REDYELRED
                }
              }
            }
          }

#
#   REM**Calculate SIG18RA
#SIG18R:
```

```
#       SIG18RA = RED
#       IF DOT3 = WBD THEN GOTO SIG16R
#       IF BK17 = OCC THEN GOTO SIG16R
#       SIG18RA = YEL
#       IF SIG20RABC = REDREDRED THEN GOTO SIG16R
#       SIG18RA = GRN
#

          set SIG18RA $RED
          if {$DOT3 != $WBD && $BK17 != $OCC} {
            set SIG18RA $YEL
            if {$SIG20RABC != $REDREDRED} {set SIG18RA $GRN}
          }


#  REM**Calculate SIG16RA
#SIG16R:
#       SIG16RA = RED
#       IF DOT3 = WBD THEN GOTO SIG14R
#       IF BK16 = OCC THEN GOTO SIG14R
#       SIG16RA = YEL
#       IF SIG18R = RED THEN GOTO SIG14R
#       SIG16RA = GRN

          set SIG16RA $RED
          if {$DOT3 != $WBD && $BK16 != $OCC} {
            set SIG16RA $YEL
            if {$SIG18RA != $RED} {set SIG16RA $GRN}
          }

#
#  REM**Calculate SIG14RA and SIG14RC
#SIG14R:
#       SIG14RA = RED: SIG14RC = RED
#       IF DOT3 = WBD THEN GOTO SIG12R
#       IF OS5 = OCC THEN GOTO SIG12R
#       IF BK15 = OCC THEN GOTO SIG12R
#       SIG14R = YEL
#       IF SIG16RA = RED THEN GOTO SIG12R
#       SIG14R = GRN
#       IF SM10 = TUN THEN SIG14RA = SIG14R ELSE SIG14RC = SIG14R

          set SIG14RA $RED; set SIG14RC $RED
          if {$DOT3 != $WBD &&
              $OS5  != $OCC &&
```

```
467                 $BK15 != $OCC} {
                  set SIG14R $YEL
                  if {$SIG16RA != $RED} {set SIG14R $GRN}
                  if {$SM10 == $TUN} {set SIG14RA $SIG14R} else {set SIG14RC $SIG14R}
                }
472
    #
    #  REM**Calculate SIG12RABC
    #SIG12R:
    #       SIG12RABC = REDREDRED
477 #       IF OS4 = OCC THEN GOTO SIG8R
    #       IF SM8 = TUR THEN GOTO SM8REV
    #       IF BK13 = OCC THEN GOTO SIG8R
    #       SIG12RABC = YELREDRED
    #       IF SIG14RA = RED THEN GOTO SIG8R
482 #       SIG12RABC = GRNREDRED: GOTO SIG8R
    #SM8REV:
    #       IF SM9 = TUR THEN SIG12RABC = REDREDYEL: GOTO SIG8R
    #       IF BK14 = OCC THEN GOTO SIG8R
    #       SIG12RABC = REDYELRED
487

                set SIG12RABC $REDREDRED
                if {$OS4 != $OCC} {
                  if {$SM8 != $TUR} {
492                 if {$BK13 != $OCC} {
                      set SIG12RABC $YELREDRED
                      if {$SIG14RA != $RED} {set SIG12RABC $GRNREDRED}
                    }
                  } else {
497                 # SM8REV:
                    if {$SM9 == $TUR} {
                      set SIG12RABC $REDREDYEL
                    } elseif {$BK14 != $OCC} {
                      set SIG12RABC $REDYELRED
502                 }
                  }
                }


507 #
    #  REM**Calculate SIG8RA and SIG8RC
    #SIG8R:
    #       SIG8RA = RED: SIG8RC = RED
    #       IF DOT2 = WBD THEN GOTO SIG6R
```

```
512 #       IF OS3 = OCC THEN GOTO SIG6R
    #       IF BK11 = OCC THEN GOTO SIG6R
    #       SIG8R = YEL
    #       IF SIG12RABC = REDREDRED THEN GOTO SIG6R
    #       SIG8R = GRN
517 #       IF SM7 = TUN THEN SIG8RA = SIG8R ELSE SIG8RC = SIG8R

              set SIG8RA $RED; set SIG8RC $RED
              if {$DOT2 != $WBD && $OS3 != $OCC && $BK11 != $OCC} {
                set SIG8R $YEL
522             if {$SIG12RABC != $REDREDRED} {set SIG8R $GRN}
                if (SM7 == TUN) {set SIG8RA $SIG8R} else {set SIG8RC $SIG8R}
              }


527 #
    #   REM**Calculate SIG6RAB
    #SIG6R:
    #       SIG6RAB = REDRED
    #       IF OS2 = OCC THEN GOTO SIG4R
532 #       IF SM6 = TUR THEN GOTO SM6REV
    #       IF BK9 = OCC THEN GOTO SIG4R
    #       SIG6RAB = YELRED
    #       IF SIG8RA = RED THEN GOTO SIG4R
    #       SIG6RAB = GRNRED: GOTO SIG4R
537 #
    #SM6REV:
    #       IF BK10 = OCC THEN GOTO SIG4R
    #       SIG6RAB = REDYEL

542           set SIG6RAB $REDRED
              if {$OS2 != $OCC} {
                if {$SM6 != $TUR} {
                  if {$BK9 != $OCC} {
                    set SIG6RAB $YELRED
547                 if {$SIG8RA != $RED} {set SIG6RAB $GRNRED}
                  }
                } else {
                  # SM6REV:
                  if {$BK10 != $OCC} {set SIG6RAB $REDYEL}
552             }
              }

    #
    #   REM**Calculate SIG4RA
```

```
#SIG4R:
#       SIG4RA = RED
#       IF DOT1 = WBD THEN GOTO SIG2R
#       IF BK8 = OCC THEN GOTO SIG2R
#       SIG4RA = YEL
#       IF SIG6RAB = REDRED THEN GOTO SIG2R
#       SIG4RA = GRN

          set SIG4RA $RED
          if {$DOT1 != $WBD && $BK8 != $OCC} {
            set SIG4RA $YEL
            if {$SIG6RAB != $REDRED} {set SIG4RA $GRN}
          }

#
#  REM**Calculate Red Butte exit signals
#  REM**Initialize all exit signals to RED
#SIG2R:
#       set SIG2RA $RED: SIG2RB = RED: SIG2RC = RED
#       SIG2RD = RED: SIG2RE = RED: SIG2RF = RED
#
#  REM**Calculate exit signal value for aligned track
#       EXITSIG = RED
#       IF DOT1 = WBD THEN GOTO SIGECMP
#       IF OS1 = OCC THEN GOTO SIGECMP
#       IF BK7 = OCC THEN GOTO SIGECMP
#       EXITSIG = YEL
#       IF SIG4RA = RED THEN GOTO EXSIG
#       EXITSIG = GRN
#
#  REM**Set track aligned exit signal to calculated signal value
#EXSIG:
#       IF TK = 1 THEN SIG2RA = EXITSIG
#       IF TK = 2 THEN SIG2RB = EXITSIG
#       IF TK = 3 THEN SIG2RC = EXITSIG
#       IF TK = 4 THEN SIG2RD = EXITSIG
#       IF TK = 5 THEN SIG2RE = EXITSIG
#       IF TK = 6 THEN SIG2RF = EXITSIG
#SIGECMP:

          set SIG2RA $RED; set SIG2RB $RED; set SIG2RC $RED
          set SIG2RD $RED; set SIG2RE $RED; set SIG2RF $RED
          set EXITSIG $RED
          if {$DOT1 != $WBD && $OS1 != $OCC && $BK7 != $OCC} {
            set EXITSIG $YEL
```

```
602             if {$SIG4RA != $RED} {set EXITSIG $GRN}
                }
                switch $TK {
                  1 {set SIG2RA $EXITSIG}
                  2 {set SIG2RB $EXITSIG}
607               3 {set SIG2RC $EXITSIG}
                  4 {set SIG2RD $EXITSIG}
                  5 {set SIG2RE $EXITSIG}
                  6 {set SIG2RF $EXITSIG}
                }
612
    #
    #  REM*****************************
    #  REM*****************************
    #  REM**CALCULATE WESTBOUND SIGNALS**
617 #  REM*****************************
    #  REM*****************************
    #
    #    REM**Calculate SIG2LAB (entrance signal into staging)
    #SIG2L:
622 #      SIG2LAB = REDRED
    #      IF OS1 = OCC THEN GOTO SIG4L
    #      IF TK = 1 AND BK1 = CLR THEN SIG2LAB = YELRED: GOTO SIG4L
    #      IF TK = 2 AND BK2 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
    #      IF TK = 3 AND BK3 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
627 #      IF TK = 4 AND BK4 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
    #      IF TK = 5 AND BK5 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
    #      IF TK = 6 AND BK6 = CLR THEN SIG2LAB = REDYEL

            set SIG2LAB $REDRED
632         if {$OS1 != $OCC} {
              if {$TK == $1 && $BK1 == $CLR} {
                set SIG2LAB $YELRED
              } elseif {$TK == $2 && $BK2 == $CLR} {
                set SIG2LAB $REDYEL
637           } elseif {$TK == $3 && $BK3 == $CLR} {
                set SIG2LAB $REDYEL
              } elseif {$TK == $4 && $BK4 == $CLR} {
                set SIG2LAB $REDYEL
              } elseif {$TK == $5 && $BK5 == $CLR} {
642             set SIG2LAB $REDYEL
              } elseif {$TK == $6 && $BK6 == $CLR} {
                set SIG2LAB $REDYEL
              }
            }
```

```
647
    #
    #  REM**Calculate SIG4LA
    #SIG4L:
    #       SIG4LA = RED
652 #       IF DOT1 = EBD THEN GOTO SIG6L
    #       IF BK7 = OCC THEN GOTO SIG6L
    #       SIG4LA = YEL
    #       IF SIG2LAB = REDRED THEN GOTO SIG6L
    #       SIG4LA = GRN
657
            set SIG4LA $RED
            if {$DOT1 != $EBD && $BK7 != $OCC} {
              set SIG4LA $YEL
              if {$SIG2LAB != $REDRED} {set SIG4LA $GRN}
662           }

    #
    #  REM**Calculate SIG6LA and SIG6LC
    #SIG6L:
667 #       SIG6LA = RED: SIG6LC = RED
    #       IF DOT1 = EBD THEN GOTO SIG8L
    #       IF OS2 = OCC THEN GOTO SIG8L
    #       IF BK8 = OCC THEN GOTO SIG8L
    #       SIG6L = YEL
672 #       IF SIG4LA = RED THEN GOTO SIG8L
    #       SIG6L = GRN
    #       IF SM6 = TUN THEN SIG6LA = SIG6L ELSE SIG6LC = SIG6L
    #

677         set SIG6LA $RED; set SIG6LC $RED
            if {$DOT1 != $EBD && $OS2 != $OCC && $BK8 != $OCC} {
              set SIG6L $YEL
              if {$SIG4LA != $RED} {set SIG6L $GRN}
              if {$SM6 == $TUN} {set SIG6LA $SIG6L} else {set SIG6LC $SIG6L}
682           }


    #  REM**Calculate SIG8LAB
    #SIG8L:
687 #       SIG8LAB = REDRED
    #       IF OS3 = OCC THEN GOTO SIG12L
    #       IF SM7 = TUR THEN GOTO SM7REV
    #       IF BK9 = OCC THEN GOTO SIG12L
    #       SIG8LAB = YELRED
```

```
692 #       IF SIG6LA = RED THEN GOTO SIG12L
    #       SIG8LAB = GRNRED: GOTO SIG12L
    #SM7REV:
    #       IF BK10 = OCC THEN GOTO SIG12L
    #       SIG8LAB = REDYEL
697
            set SIG8LAB $REDRED
            if {$OS3 != $OCC} {
              if {$SM7 != $TUR} {
                if {$BK9 != $OCC} {
702               set SIG8LAB $YELRED
                  if {$SIG6LA != $RED} {set SIG8LAB $GRNRED}
                }
              } else {
                # SM7REV:
707             if {$BK10 != $OCC} {set SIG8LAB $REDYEL}
              }
            }

    #
712 #  REM**Calculate SIG12LA, SIG12LC and SIG12LD
    #SIG12L:
    #       SIG12LA = RED: SIG12LC = RED: SIG12LD = RED
    #       IF DOT2 = EBD THEN GOTO SIG14L
    #       IF OS4 = OCC THEN GOTO SIG14L
717 #       IF BK11 = OCC THEN GOTO SIG14L
    #       SIG12L = YEL
    #       IF SIG8LAB = REDRED THEN GOTO SIG14L
    #       SIG12L = GRN
    #       IF SM8 = TUN THEN SIG12LA = SIG12L: GOTO SIG14L
722 #       IF SM9 = TUN THEN SIG12LC = SIG12L ELSE SIG12LD = SIG12L

            set SIG12LA $RED; set SIG12LC $RED; set SIG12LD $RED
            if {$DOT2 != $EBD && $OS4 != $OCC && $BK11 != $OCC} {
              set SIG12L $YEL
727           if {$SIG8LAB != $REDRED} {set SIG12L $GRN}
              if {$SM8 == $TUN} {
                set SIG12LA $SIG12L
              } elseif {$SM9 == $TUN} {
                set SIG12LC $SIG12L
732           } else {
                set SIG12LD $SIG12L
              }
            }
```

```
737
    #
    #  REM**Calculate SIG14LAB
    #SIG14L:
    #       SIG14LAB = REDRED
742 #       IF OS5 = OCC THEN GOTO SIG16L
    #       IF SM10 = TUR THEN GOTO SM10REV
    #       IF BK13 = OCC THEN GOTO SIG16L
    #       SIG14LAB = YELRED
    #       IF SIG12LA = RED THEN GOTO SIG16L
747 #       SIG14LAB = GRNRED: GOTO SIG16L
    #SM10REV:
    #       IF BK14 = OCC THEN GOTO SIG16L
    #       SIG14LAB = REDYEL

752           set SIG14LAB $REDRED
              if {$OS5 != $OCC} {
                if {$SM10 != $TUR} {
                  if {$BK13 != $OCC} {
                    set SIG14LAB $YELRED
757                 if {$SIG12LA != $RED} {set SIG14LAB $GRNRED}
                  }
                } else {
                  # SM10REV
                  if {$BK14 != $OCC} {set SIG14LAB $REDYEL}
762             }
              }

    #
    #  REM**Calculate SIG16LA
767 #SIG16L:
    #       SIG16LA = RED
    #       IF DOT3 = EBD THEN GOTO SIG18L
    #       IF BK15 = OCC THEN GOTO SIG18L
    #       SIG16LA = YEL
772 #       IF SIG14LAB = REDRED THEN GOTO SIG18L
    #       SIG16LA = GRN

              set SIG16LA $RED
              if {$DOT3 != $EBD && $BK15 != $OCC} {
777             set SIG16LA $YEL
                if {$SIG14LAB != $REDRED} {set SIG16LA $GRN}
              }

    #
```

```
#  REM**Calculate SIG18LA
#SIG18L:
#       SIG18LA = RED
#       IF DOT3 = EBD THEN GOTO SIG20L
#       IF BK16 = OCC THEN GOTO SIG20L
#       SIG18LA = YEL
#       IF SIG16LA = RED THEN GOTO SIG20L
#       SIG18LA = GRN

          set SIG18LA $RED
          if {$DOT3 != $EBD && $BK16 != $OCC} {
            set SIG18LA $YEL
            if {$SIG16LA != $RED} {set SIG18LA $GRN}
          }

#
#  REM**Calculate SIG20LA, SIG20LB and SIG20LD
#SIG20L:
#       SIG20LA = RED: SIG20LB = RED: SIG20LD = RED
#       IF DOT3 = EBD THEN GOTO SIGWCMP
#       IF OS6 = OCC THEN GOTO SIGWCMP
#       IF BK17 = OCC THEN GOTO SIGWCMP
#       SIG20L = YEL
#       IF SIG18LA <> RED THEN SIG20L = GRN
#       IF SM11 = TUR THEN SIG20LB = SIG20L: GOTO SIGWCMP
#       IF TF12 = TGR THEN SIG20LD = SIG20L ELSE SIG20LA = SIG20L
#SIGWCMP:

          set SIG20LA $RED; set SIG20LB $RED; set SIG20LD $RED
          if {$DOT3 != $EBD && $OS6 != $OCC && $BK17 != $OCC} {
            set SIG20L $YEL
            if {$SIG18LA != $RED} {set SIG20L $GRN}
            if {$SM11 == $TUR} {
              set SIG20LB $SIG20L
            } elseif {$TF12 == $TGR} {
              set SIG20LD $SIG20L
            } else {
              set SIG20LA $SIG20L
            }
          }


#
#  REM**PACK AND WRITE OUTPUT BYTES
#       OB(1) = SIG2LAB                    'CARD 0 PORT A
```

```
827 #       OB(1) = SIG2RA * B5 OR OB(1)

              set Outputs [expr {$SIG2LAB | \
                                 $SIG2RA << $B5}];      # CARD 0 PORT A

832 #
    #       OB(2) = SIG2RB                    'CARD 0 PORT B
    #       OB(2) = SIG2RC * B3 OR OB(2)
    #       OB(2) = SM1 * B6 OR OB(2)
    #       OB(2) = SM2 * B7 OR OB(2)
837
              lappend Outputs [expr {$SIG2RB | \
                                     $SIG2RC << $B3 | \
                                     $SM1    << $B6 | \
                                     $SM2    << $B7}];          #CARD 0 PORT B
842 #
    #       OB(3) = SIG2RD                    'CARD 0 PORT C
    #       OB(3) = SIG2RE * B3 OR OB(3)
    #       OB(3) = SM3 * B6 OR OB(3)
    #       OB(3) = SM4 * B7 OR OB(3)
847
              lappend Outputs [expr {$SIG2RD | \
                                     $SIG2RE << $B3 | \
                                     $SM3    << $B6 | \
                                     $SM4    << $B7}];          #CARD 0 PORT C
852 #
    #       OB(4) = SIG2RF                    'CARD 0 PORT D
    #       OB(4) = SIG4LA * B3 OR OB(4)
    #       OB(4) = SM5 * B6 OR OB(4)
    #       OB(4) = SM6 * B7 OR OB(4)
857
              lappend Outputs [expr {$SIG2RF | \
                                     $SIG4LA << $B3 | \
                                     $SM5    << $B6 | \
                                     $SM6    << $B7}];          #CARD 0 PORT D
862 #
    #       OB(5) = SIG6RAB                   'CARD 3 PORT A
    #       OB(5) = SIG4RA * B5 OR OB(5)

              lappend Outputs [expr {$SIG6RAB | \
867                                  $SIG4RA << $B5}];          #CARD 3 PORT A
    #
    #       OB(6) = SIG6LA                    'CARD 3 PORT B
    #       OB(6) = SIG6LC * B3 OR OB(6)
    #       OB(6) = SM7 * B6 OR OB(6)
```

```
872  #      OB(6) = SM8 * B7 OR OB(6)

              lappend Outputs [expr {$SIG6LA | \
                                     $SIG6LC << $B3 | \
                                     $SM7    << $B6 | \
877                                  $SM8    << $B7}];          #CARD 3 PORT B
     #
     #      OB(7) = SIG8LAB                    'CARD 3 PORT C
     #      OB(7) = SIG8RA * B5 OR OB(7)

882           lappend Outputs [expr {$SIG8LAB | \
                                     $SIG8RA << $B5}];          #CARD 3 PORT C
     #
     #      OB(8) = SIG8RC                     'CARD 3 PORT D
     #      OB(8) = SIG12LA * B3 OR OB(8)
887
              lappend Outputs [expr {$SIG8RC | \
                                     $SIG12LA << $B3}];         #CARD 3 PORT D
     #
     #      OB(9) = SIG12RABC                  'CARD 4 PORT A
892  #      OB(9) = SM9 * B7 OR OB(9)

              lappend Outputs [expr {$SIG12RABC | \
                                     $SM9 << $B7}];     #CARD 4 PORT A
     #
897  #      OB(10) = SIG12LC                   'CARD 4 PORT B
     #      OB(10) = SIG12LD * B3 OR OB(10)
     #      OB(10) = SM10 * B6 OR OB(10)
     #      OB(10) = SM11 * B7 OR OB(10)

902           lappend Outputs [expr {$SIG12LC | \
                                     $SIG12LD << $B3 | \
                                     $SM10    << $B6 | \
                                     $SM11    << $B7}];         #CARD 4 PORT B
     #
907  #      OB(11) = SIG14LAB                  'CARD 4 PORT C
     #      OB(11) = SIG14RA * B5 OR OB(11)

              lappend Outputs [expr {$SIG14LAB | \
                                     $SIG14RA << $B5}];         #CARD 4 PORT C
912  #
     #      OB(12) = SIG14RC                   'CARD 4 PORT D
     #      OB(12) = SIG16LA * B3 OR OB(12)

              lappend Outputs [expr {$SIG14RC | \
```

```
                                    $SIG16LA << $B3}];        #CARD 4 PORT D
#
#       OB(13) = SIG16RA                    'CARD 5 PORT A
#       OB(13) = SIG18LA * B3 OR OB(13)

          lappend Outputs [expr {$SIG16RA | \
                                    $SIG18LA << $B3}];        #CARD 5 PORT A
#
#       OB(14) = SIG18RA                    'CARD 5 PORT B
#       OB(14) = SIG20LA * B3 OR OB(14)

          lappend Outputs [expr {$SIG18RA | \
                                    $SIG20LA << $B3}];        #CARD 5 PORT B
#
#       OB(15) = SIG20RABC                  'CARD 5 PORT C

          lappend Outputs $SIG20RABC; #CARD 5 PORT C
#
#       OB(16) = SIG20LB                    'CARD 5 PORT D
#       OB(16) = SIG20LD * B3 OR OB(16)

          lappend Outputs [expr {$SIG20LB | \
                                    $SIG20LD << $B3}];        #CARD 5 PORT D
#
#       GOSUB OUTPUTS       'INVOKE OUTPUTS SUBROUTINE

          if {[catch {bus Outputs $Outputs $UA} result]} {
                  # Handle error.
                  puts -nonewline stderr "Could not write to the output ports of "
                  puts stderr "SUISC card at UA $UA: $result"
                  break
          }

#
#   REM**RETURN TO BEGINNING OF REAL-TIME LOOP
#       GOTO BRTL

}
```

5.4.3 From Chapter 14: Distributed Application Examples

Listing 5.6: Figure 14-5, Tcl Translation

```
# $Id: fig14-5.tcl 1465 2013-03-17 15:24:03Z heller $

#       DEFINT A-Z
```

```
#       DECLARE SUB INIT ()
#       DECLARE SUB OUTPUTS ()
#       DECLARE SUB INPUTS ()
#       DECLARE SUB RXBYTE ()
#       DECLARE SUB TXPACK ()
#   REM**MULTI-NODE EXAMPLE USING 3-ASPECT COLOR LIGHT SIGNALS**
#   REM**GLOBALIZE SERIAL PROTOCOL HANDLING VARIABLES
#       DIM SHARED OB(60), IB(60), CT(15), TB(80)
#       COMMON SHARED UA, COMPORT, BAUD100, NDP$, DL, NS, NI, NO, MAXTRIES
#       COMMON SHARED INBYTE, ABORTIN, INTRIES, INITERR, PA, LM, MT

# Load MRR System packages
# Add MRR System package Paths
lappend auto_path /usr/local/share/MRRSystem;# Tcl packages
package require Cmri 2.0.0;#            Load the CMR/I package

#
#   REM**INITIALIZE CONSTANTS FOR PACKING AND UNPACKING I/O BYTES
#       B0 = 1: B1 = 2: B2 = 4: B3 = 8: B4 = 16: B5 = 32: B6 = 64: B7 = 128
#       W1 = 1: W2 = 3: W3 = 7: W4 = 15: W5 = 31: W6 = 63: W7 = 127

#   REM**SUSIC SINGLE-NODE SYSTEM USING 3-ASPECT COLOR LIGHT SIGNALS**
#   REM**DEFINE VARIABLE TYPES AND ARRAY SIZES
#       DEFINT A-Z
#         DIM OB(60), IB(60), TB(80), CT(15)
#
#   REM**DEFINE CONSTANTS FOR PACKING AND UNPACKING I/O BYTES
#       B0 = 1: B1 = 2: B2 = 4: B3 = 8: B4 = 16: B5 = 32: B6 = 64: B7 = 128
#       W1 = 1: W2 = 3: W3 = 7: W4 = 15: W5 = 31: W6 = 63: W7 = 127

# Bit shift constants
#
set B0 0
set B1 1
set B2 2
set B3 3
set B4 4
set B5 5
set B6 6
set B7 7

# Bit mask constants
#
set W1 0x01
set W2 0x03
```

```
    set W3 0x07
50  set W4 0x0f
    set W5 0x1f
    set W6 0x3f
    set W7 0x7f

55  #
    #  REM**DEFINE BLOCK OCCUPATION CONSTANTS
    #      CLR = 0          'Clear
    #      OCC = 1          'Occupied

60  set CLR 0
    set OCC 1

    #
    #     REM**DEFINE SIGNAL ASPECTS - Single Head
65  #      DRK = 0    'Dark     000
    #      GRN = 1    'Green    001
    #      YEL = 2    'Yellow   010
    #      RED = 4    'Red      100

70  set DRK 0;# Dark     000
    set GRN 1;# Green    001
    set YEL 2;# Yellow   010
    set RED 4;# Red      100

75  #
    #  REM**DEFINE SIGNAL ASPECTS - Double Head
    #      GRNRED = 17    'Green over red     10001
    #      YELRED = 18    'Yellow over red    10010
    #      REDYEL = 12    'Red over yellow    01100
80  #      REDRED = 20    'Red over red       10100

    set GRNRED 0x11;# Green over red     10001
    set YELRED 0x12;# Yellow over red    10010
    set REDYEL 0x0c;# Red over yellow    01100
85  set REDRED 0x14;# Red over red       10100

    #
    #  REM**DEFINE SIGNAL ASPECTS - Triple Head
    #      REDREDRED = 84    'Red over red over red      1010100
90  #      REDREDYEL = 52    'Red over red over yellow   0110100
    #      REDYELRED = 76    'Red over yellow over red   1001100
    #      YELREDRED = 82    'Yellow over red over red   1010010
    #      GRNREDRED = 81    'Green over red over red    1010001
```

```
95  set REDREDRED 0x54;# Red over red over red      1010100
    set REDREDYEL 0x34;# Red over red over yellow   0110100
    set REDYELRED 0x4c;# Red over yellow over red   1001100
    set YELREDRED 0x52;# Yellow over red over red   1010010
    set GRNREDRED 0x51;# Green over red over red    1010001
100
    #
    #  REM**INITIALIZE TURNOUT POSITION CONSTANTS
    #  REM**Constants assume direct connection to output pins (not using SMC)
    #  REM**See other examples for using SMC card with one I/O line per machine
105 #     TUN = 1      '01
    #     TUR = 2      '10

    set TUN 0
    set TUR 1
110
    #
    #  REM**DEFINE PUSHBUTTON AND TOGGLE POSITIONS
    #     PBP = 1    'Pushbutton pressed
    #     TGR = 1    'Toggle reverse position
115 #     TGN = 0    'Toggle normal position

    set PBP 1;# Pushbutton pressed
    set TGR 1;# Toggle reverse position
    set TGN 0;# Toggle normal position
120
    #
    #  REM**DEFINE DIRECTION-OF-TRAFFIC RUNNING ON SINGLE TRACK
    #     NDT = 0       'No direction-of-traffic defined on single track
    #     EBD = 1       'EastBounD Direction
125 #     WBD = 2       'WestBounD Direction

    set NDT 0;# No direction-of-traffic defined on single track
    set EBD 1;# EastBounD Direction
    set WBD 2;# WestBounD Direction
130
    #
    #  REM**INITIALIZE DIRECTION-OF-TRAFFIC VARIABLES TO NO DIRECTION-OF-TRAFFIC
    #     DOT1 = NDT: DOT2 = NDT: DOT3 = NDT
          set DOT1 $NDT; set DOT2 $NDT; set DOT3 $NDT
135
    #
    #  PRINT "MULTIPLE NODE EXAMPLE USING 3-ASPECT COLOR LIGHT SIGNALS"
```

```
        puts "MULTIPLE NODE RAILROAD EXAMPLE USING 3-ASPECT COLOR LIGHT SIGNALS"
140
    #
    #  REM**INITIALIZE GENERAL SERIAL PROTOCOL PARAMETERS (same for all nodes)
    #     COMPORT = 3       'PC SERIAL COMMUNICATIONS PORT = 1, 2, 3 OR 4
    #     BAUD100 = 192     'BAUD RATE OF 19200 DIVIDED BY 100
145 #     DL = 0            'USIC TRANSMISSION DELAY
    #     MAXTRIES = 10000  'MAXIMUM READ TRIES BEFORE ABORT INPUTS
    # Connect to the bus on COM3: (/dev/ttyS2), at 19200 BAUD, with
    # a retry count of 10000, capturing error messages.
    if {[catch {cmri::CMri bus /dev/ttyS2 -baud 19200 -retries 10000} result]} {
150         # Handle error.
            puts -nonewline stderr "Could not connect to CMR/I bus on /dev/ttyS2: "
            puts stderr "$result"
            rename bus {}
            exit 99
155 }
    #
    #  REM**INITIALIZE NODE 0 (SMINI)
    #     UA = 0            'USIC NODE ADDRESS
    #     NDP$ = "M"        'NODE DEFINITION PARAMETER
160 #     NS = 0            'NUMBER OF 2-LEAD SEARCHLIGHT SIGNALS
    #     NI = 3               'NUMBER OF INPUT PORTS
    #     NO = 6               'NUMBER OF OUTPUT PORTS
    #     CALL INIT         'INVOKE INITIALIZATION SUBROUTINE
    set UA 0
165 if {[catch {bus InitBoard {} 3 6 0 $UA SMINI 0} result]} {
            # Handle error.
            puts -nonewline stderr "Could not initialize SMINI card at UA "
            puts stderr "$UA: $result"
            rename bus {}
170         exit 99
    }
    #
    #  REM**INITIALIZE NODE 1 (SMINI)
    #     UA = 1            'USIC NODE ADDRESS
175 #     NDP$ = "M"        'NODE DEFINITION PARAMETER
    #     NS = 0            'NUMBER OF 2-LEAD SEARCHLIGHT SIGNALS
    #     NI = 3               'NUMBER OF INPUT PORTS
    #     NO = 6               'NUMBER OF OUTPUT PORTS
    #     CALL INIT         'INVOKE INITIALIZATION SUBROUTINE
180 set UA 1
    if {[catch {bus InitBoard {} 3 6 0 $UA SMINI 0} result]} {
            # Handle error.
            puts -nonewline stderr "Could not initialize SMINI card at UA "
```

```
            puts stderr "$UA: $result"
            rename bus {}
            exit 99
        }
        #
        #   REM**INITIALIZE NODE 2 (SMINI)
        #       UA = 2              'USIC NODE ADDRESS
        #       NDP$ = "M"          'NODE DEFINITION PARAMETER
        #       NS = 0              'NUMBER OF 2-LEAD SEARCHLIGHT SIGNALS
        #       NI = 3                  'NUMBER OF INPUT PORTS
        #       NO = 6                  'NUMBER OF OUTPUT PORTS
        #       CALL INIT           'INVOKE INITIALIZATION SUBROUTINE
        set UA 2
        if {[catch {bus InitBoard {} 3 6 0 $UA SMINI 0} result]} {
            # Handle error.
            puts -nonewline stderr "Could not initialize SMINI card at UA "
            puts stderr "$UA: $result"
            rename bus {}
            exit 99
        }
        #
        #   REM**INITIALIZE NODE 3 (SMINI)
        #       UA = 3              'USIC NODE ADDRESS
        #       NDP$ = "M"          'NODE DEFINITION PARAMETER
        #       NS = 0              'NUMBER OF 2-LEAD SEARCHLIGHT SIGNALS
        #       NI = 3                  'NUMBER OF INPUT PORTS
        #       NO = 6                  'NUMBER OF OUTPUT PORTS
        #       CALL INIT           'INVOKE INITIALIZATION SUBROUTINE
        set UA 3
        if {[catch {bus InitBoard {} 3 6 0 $UA SMINI 0} result]} {
            # Handle error.
            puts -nonewline stderr "Could not initialize SMINI card at UA "
            puts stderr "$UA: $result"
            rename bus {}
            exit 99
        }
        #
        #   REM**INITIALIZE NODE 4 (SUSIC)
        #       UA = 4              'USIC NODE ADDRESS
        #       NDP$ = "X"          'NODE DEFINITION PARAMETER
        #       NS = 1              'NUMBER OF CARD SETS OF 4
        #       CT(1) = 6           'CARD TYPE ARRAY ELEMENT FOR GROUPING OF OIXX
        #       NI = 4                  'NUMBER OF INPUT PORTS
        #       NO = 4                  'NUMBER OF OUTPUT PORTS
        #       CALL INIT           'INVOKE INITIALIZATION SUBROUTINE
```

```
    set UA 4
230 if {[catch {bus InitBoard {0x06} 4 4 0 $UA SUSICI 0} result]} {
        # Handle error.
        puts -nonewline stderr "Could not initialize SUSIC card at UA "
        puts stderr "$UA: $result"
        rename bus {}
235     exit 99
    }
    #
    #BRTL: '*******BEGIN REAL TIME LOOP*******

240 while {true} {

    #  REM**READ AND UNPACK INPUTS FOR NODE 0 (SMINI)
    #     UA = 0: NI = 3
    #     CALL INPUTS
245 #     BK1 = IB(1)\B0 AND W1    'NODE 0 CARD 2 PORT A
    #     BK2 = IB(1)\B1 AND W1
    #     BK3 = IB(1)\B2 AND W1
    #     BK4 = IB(1)\B3 AND W1
    #     BK5 = IB(1)\B4 AND W1
250 #     BK6 = IB(1)\B5 AND W1
    #     OS1 = IB(1)\B6 AND W1
    #     BK7 = IB(1)\B7 AND W1

        set UA 0
255     if {[catch {bus Inputs 3 $UA} result]} {
            puts -nonewline stderr "Could not read from the input ports of "
            puts stderr "SMINI card at UA $UA: $result"
            break
        }
260     set Inputs $result
        set tempByte [lindex $Inputs 0];#        NODE 0 CARD 2 PORT A
        set BK1 [expr {$tempByte >> $B0 & $W1}]
        set BK2 [expr {$tempByte >> $B1 & $W1}]
        set BK3 [expr {$tempByte >> $B2 & $W1}]
265     set BK4 [expr {$tempByte >> $B3 & $W1}]
        set BK5 [expr {$tempByte >> $B4 & $W1}]
        set BK6 [expr {$tempByte >> $B5 & $W1}]
        set OS1 [expr {$tempByte >> $B6 & $W1}]
        set BK7 [expr {$tempByte >> $B7 & $W1}]
270

    #
    #  REM**READ AND UNPACK INPUTS FOR NODE 1 (SMINI)
```

```
#       UA = 1: NI = 3
#       CALL INPUTS
#       BK8 = IB(1)\B0 AND W1       'NODE 1 CARD 2 PORT A
#       OS2 = IB(1)\B1 AND W1
#       BK9 = IB(1)\B2 AND W1
#       BK10 = IB(1)\B3 AND W1
#       OS3 = IB(1)\B4 AND W1
#       BK11 = IB(1)\B5 AND W1

          set UA 1
          if {[catch {bus Inputs 3 $UA} result]} {
                puts -nonewline stderr "Could not read from the input ports of "
                puts stderr "SMINI card at UA $UA: $result"
                break
          }
          set Inputs $result
          set tempByte [lindex $Inputs 0];#          NODE 1 CARD 2 PORT A
          set BK8  [expr {$tempByte >> $B0 & $W1}]
          set OS2  [expr {$tempByte >> $B1 & $W1}]
          set BK9  [expr {$tempByte >> $B2 & $W1}]
          set BK10 [expr {$tempByte >> $B3 & $W1}]
          set OS3  [expr {$tempByte >> $B4 & $W1}]
          set BK11 [expr {$tempByte >> $B5 & $W1}]

#
# REM**READ AND UNPACK INPUTS FOR NODE 2 (SMINI)
#       UA = 2: NI = 3
#       CALL INPUTS
#       OS4 = IB(1)\B0 AND W1       'NODE 2 CARD 2 PORT A
#       BK13 = IB(1)\B1 AND W1
#       BK14 = IB(1)\B2 AND W1
#       OS5 = IB(1)\B3 AND W1
#       BK15 = IB(1)\B4 AND W1
#       BK20 = IB(1)\B5 AND W1

          set UA 2
          if {[catch {bus Inputs 3 $UA} result]} {
                puts -nonewline stderr "Could not read from the input ports of "
                puts stderr "SMINI card at UA $UA: $result"
                break
          }
          set Inputs $result
          set tempByte [lindex $Inputs 0];#          NODE 2 CARD 2 PORT A
          set OS4  [expr {$tempByte >> $B0 & $W1}]
          set BK13 [expr {$tempByte >> $B1 & $W1}]
```

```
            set BK14 [expr {$tempByte >> $B2 & $W1}]
320         set OS5  [expr {$tempByte >> $B3 & $W1}]
            set BK15 [expr {$tempByte >> $B4 & $W1}]
            set BK20 [expr {$tempByte >> $B5 & $W1}]
    #
    #  REM**READ AND UNPACK INPUTS FOR NODE 3 (SMINI)
325 #      UA = 3: NI = 3
    #      CALL INPUTS
    #      BK16 = IB(1)\B0 AND W1      'NODE 3 CARD 2 PORT A
    #      BK17 = IB(1)\B1 AND W1
    #      OS6 = IB(1)\B2 AND W1
330 #      BK18 = IB(1)\B3 AND W1
    #      TF12 = IB(1)\B4 AND W1

            set UA 3
            if {[catch {bus Inputs 3 $UA} result]} {
335                 puts -nonewline stderr "Could not read from the input ports of "
                    puts stderr "SMINI card at UA $UA: $result"
                    break
            }
            set Inputs $result
340         set tempByte [lindex $Inputs 0];#         NODE 3 CARD 2 PORT A
            set BK16 [expr {$tempByte >> $B0 & $W1}]
            set BK17 [expr {$tempByte >> $B1 & $W1}]
            set OS6  [expr {$tempByte >> $B2 & $W1}]
            set BK18 [expr {$tempByte >> $B3 & $W1}]
345         set TF12 [expr {$tempByte >> $B4 & $W1}]
    #
    #  REM**READ AND UNPACK INPUTS FOR NODE 4 (SUSIC)
    #      UA = 4: NI = 4
    #      CALL INPUTS
350 #      PB1 = IB(1)\B0 AND W1       'NODE 4 CARD 1 PORT A
    #      PB2 = IB(1)\B1 AND W1
    #      PB3 = IB(1)\B2 AND W1
    #      PB4 = IB(1)\B3 AND W1
    #      PB5 = IB(1)\B4 AND W1
355 #      PB6 = IB(1)\B5 AND W1
    #      TG6 = IB(1)\B6 AND W1
    #      TG7 = IB(1)\B7 AND W1
    #
    #      TG8 = IB(2)\B0 AND W1       'NODE 4 CARD 1 PORT B
360 #      TG9 = IB(2)\B1 AND W1
    #      TG10 = IB(2)\B2 AND W1
    #      TG11 = IB(2)\B3 AND W1
```

```
        set UA 4
        if {[catch {bus Inputs 4 $UA} result]} {
                puts -nonewline stderr "Could not read from the input ports of "
                puts stderr "SUSIC card at UA $UA: $result"
                break
        }
        set Inputs $result
        set tempByte [lindex $Inputs 0];#          NODE 4 CARD 1 PORT A
        set PB1  [expr {$tempByte >> $B0 & $W1}]
        set PB2  [expr {$tempByte >> $B1 & $W1}]
        set PB3  [expr {$tempByte >> $B2 & $W1}]
        set PB4  [expr {$tempByte >> $B3 & $W1}]
        set PB5  [expr {$tempByte >> $B4 & $W1}]
        set PB6  [expr {$tempByte >> $B5 & $W1}]
        set TG6  [expr {$tempByte >> $B6 & $W1}]
        set TG7  [expr {$tempByte >> $B7 & $W1}]
        set tempByte [lindex $Inputs 0];#          NODE 4 CARD 1 PORT B
        set TG8  [expr {$tempByte >> $B0 & $W1}]
        set TG9  [expr {$tempByte >> $B1 & $W1}]
        set TG10 [expr {$tempByte >> $B2 & $W1}]
        set TG11 [expr {$tempByte >> $B3 & $W1}]
#
#  REM**ALIGN PUSHBUTTON CONTROLLED TURNOUTS IN BUTTE IF OS1 IS CLEAR
#     IF OS1 = OCC THEN GOTO TOGGLES
#     IF PB1 = PBP THEN SM4 = TUN: SM5 = TUN: TK = 1: GOTO TOGGLES
#     IF PB2 = PBP THEN SM4 = TUR: SM5 = TUN: TK = 2: GOTO TOGGLES
#     IF PB3 = PBP THEN SM5 = TUR: SM3 = TUR: TK = 3: GOTO TOGGLES
#     IF PB4 = PBP THEN SM5 = TUR: SM3 = TUN: SM2 = TUN: TK = 4: GOTO TOGGLES
#     IF PB5 = PBP THEN
#       SM5 = TUR: SM3 = TUN: SM2 = TUR: SM1 = TUR: TK = 5: GOTO TOGGLES
#     END IF
#     IF PB6 = PBP THEN SM5 = TUR: SM3 = TUN: SM2 = TUR: SM1 = TUN: TK = 6

        if {$OS1 != $OCC} {
          if {$PB1 == $PBP} {
            set SM4 $TUN; set SM5 $TUN; set TK 1
          } elseif {$PB2 == $PBP} {
            set SM4 $TUR; set SM5 $TUN; set TK 2
          } elseif {$PB3 == $PBP} {
            set SM5 $TUR; set SM3 $TUR; set TK 3
          } elseif {$PB4 == $PBP} {
            set SM5 $TUR; set SM3 $TUN; set SM2 $TUN; set TK 4
          } elseif {$PB5 == $PBP} {
            set SM5 $TUR; set SM3 $TUN; set SM2 $TUR; set SM1 $TUR; set TK 5
          } elseif {$PB6 == $PBP} {
```

```
                set SM5 $TUR; set SM3 $TUN; set SM2 $TUR; set SM1 $TUN; set TK 6
410           }
            }


    #
415 #  REM**ALIGN TOGGLE CONTROLLED TURNOUTS IF OS SECTIONS ARE CLEAR
    #TOGGLES:
    #       IF OS2 = CLR THEN SM6 = TG6
    #       IF OS3 = CLR THEN SM7 = TG7
    #       IF OS4 = CLR THEN SM8 = TG8: SM9 = TG9
420 #       IF OS5 = CLR THEN SM10 = TG10
    #       IF OS6 = CLR THEN SM11 = TG11

            if {$OS2 == $CLR} {set SM6 $TG6}
            if {$OS3 == $CLR} {set SM7 $TG7}
425         if {$OS4 == $CLR} {set SM8 $TG8; set SM9 $TG9}
            if {$OS5 == $CLR} {set SM10 $TG10}
            if {$OS6 == $CLR} {set SM11 $TG11}

    #
430 #  REM**WHEN AN OS SECTION BECOMES OCCUPIED AND OPPOSING DIRECTION-
    #  REM**OF-TRAFFIC IS NOT LOCKED-IN THEN SET DIRECTION-OF-TRAFFIC
    #  REM**FOR SECTION OF SINGLE TRACK TO DIRECTION OF MOVEMENT
    #       IF OS1 = OCC AND DOT1 <> WBD THEN DOT1 = EBD
    #       IF OS2 = OCC AND DOT1 <> EBD THEN DOT1 = WBD
435 #       IF OS3 = OCC AND DOT2 <> WBD THEN DOT2 = EBD
    #       IF OS4 = OCC AND DOT2 <> EBD THEN DOT2 = WBD
    #       IF OS5 = OCC AND DOT3 <> WBD THEN DOT3 = EBD
    #       IF OS6 = OCC AND DOT3 <> EBD THEN DOT3 = WBD

440         if {$OS1 == $OCC && $DOT1 != $WBD} {set DOT1 $EBD}
            if {$OS2 == $OCC && $DOT1 != $EBD} {set DOT1 $WBD}
            if {$OS3 == $OCC && $DOT2 != $WBD} {set DOT2 $EBD}
            if {$OS4 == $OCC && $DOT2 != $EBD} {set DOT2 $WBD}
            if {$OS5 == $OCC && $DOT3 != $WBD} {set DOT3 $EBD}
445         if {$OS6 == $OCC && $DOT3 != $EBD} {set DOT3 $WBD}

    #  REM**RETAIN DIRECTIONT-OF-TRAFFIC WHILE SINGLE TRACK IS OCCUPIED
    #DT1: IF OS1 = OCC THEN GOTO DT2
    #       IF BK7 = OCC THEN GOTO DT2
450 #       IF BK8 = OCC THEN GOTO DT2
    #       IF OS2 = OCC THEN GOTO DT2
    #       DOT1 = NDT  'All trackage clear so set DOT1 to no direction-of-traffic
```

```
            if {$OS1 != $OCC &&
                $BK7 != $OCC &&
                $BK8 != $OCC &&
                $OS2 != $OCC} {
              set DOT1 $NDT; #All trackage clear so set DOT1 to no direction-of-traf
            }

    #
    #DT2: IF OS3 = OCC THEN GOTO DT3
    #     IF BK11 = OCC THEN GOTO DT3
    #     IF OS4 = OCC THEN GOTO DT3
    #     DOT2 = NDT  'All trackage clear so set DOT2 to no direction-of-traffic

            if {$OS3 != $OCC &&
                $BK11 != $OCC &&
                $OS4 != $OCC} {
              set DOT2 $NDT; #All trackage clear so set DOT2 to no direction-of-traf
            }

    #
    #DT3: IF OS5 = OCC THEN GOTO DOTCMP
    #     IF BK15 = OCC THEN GOTO DOTCMP
    #     IF BK16 = OCC THEN GOTO DOTCMP
    #     IF BK17 = OCC THEN GOTO DOTCMP
    #     IF OS6 = OCC THEN GOTO DOTCMP
    #     DOT1 = NDT  'All trackage clear so set DOT3 to no direction-of-traffic

            if {$OS5 != $OCC &&
                $BK15 != $OCC &&
                $BK16 != $OCC &&
                $BK17 != $OCC &&
                $OS6 != $OCC} {
              set DOT3 $NDT; #All trackage clear so set DOT3 to no direction-of-traf
            }

    #DOTCMP:
    #
    #  REM*****************************
    #  REM*****************************
    #  REM**CALCULATE EASTBOUND SIGNALS**
    #  REM*****************************
    #  REM*****************************
    #
    #  REM**Calculate SIG20RABC
    #SIG20R:
```

```
#       SIG20RABC = REDREDRED
#       IF OS6 = OCC THEN GOTO SIG18R
#       IF SM11 = TUN AND TF12 = TGR THEN SIG20RABC = REDREDYEL: GOTO SIG18R
#       IF BK18 = OCC THEN GOTO SIG18R
#       IF SM11 = TUN THEN SIG20RABC = YELREDRED ELSE SIG20RABC = REDYELRED

          set SIG20RABC $REDREDRED
          if {$OS6 != $OCC} {
            if {$SM11 == $TUN && $TF12 == TGR} {
              set SIG20RABC $REDREDYEL
            } else {
              if {$BK18 != $OCC} {
                if (SM11 == TUN) {
                  set SIG20RABC $YELREDRED
                } else {
                  set SIG20RABC $REDYELRED
                }
              }
            }
          }

#
#   REM**Calculate SIG18RA
#SIG18R:
#       SIG18RA = RED
#       IF DOT3 = WBD THEN GOTO SIG16R
#       IF BK17 = OCC THEN GOTO SIG16R
#       SIG18RA = YEL
#       IF SIG20RABC = REDREDRED THEN GOTO SIG16R
#       SIG18RA = GRN
#

          set SIG18RA $RED
          if {$DOT3 != $WBD && $BK17 != $OCC} {
            set SIG18RA $YEL
            if {$SIG20RABC != $REDREDRED} {set SIG18RA $GRN}
          }


#   REM**Calculate SIG16RA
#SIG16R:
#       SIG16RA = RED
#       IF DOT3 = WBD THEN GOTO SIG14R
#       IF BK16 = OCC THEN GOTO SIG14R
#       SIG16RA = YEL
```

```
    #       IF SIG18R = RED THEN GOTO SIG14R
545 #       SIG16RA = GRN

              set SIG16RA $RED
              if {$DOT3 != $WBD && $BK16 != $OCC} {
                set SIG16RA $YEL
550             if {$SIG18RA != $RED} {set SIG16RA $GRN}
              }

    #
    #   REM**Calculate SIG14RA and SIG14RC
555 #SIG14R:
    #       SIG14RA = RED: SIG14RC = RED
    #       IF DOT3 = WBD THEN GOTO SIG12R
    #       IF OS5 = OCC THEN GOTO SIG12R
    #       IF BK15 = OCC THEN GOTO SIG12R
560 #       SIG14R = YEL
    #       IF SIG16RA = RED THEN GOTO SIG12R
    #       SIG14R = GRN
    #       IF SM10 = TUN THEN SIG14RA = SIG14R ELSE SIG14RC = SIG14R

565           set SIG14RA $RED; set SIG14RC $RED
              if {$DOT3 != $WBD &&
                  $OS5  != $OCC &&
                  $BK15 != $OCC} {
                set SIG14R $YEL
570             if {$SIG16RA != $RED} {set SIG14R $GRN}
                if {$SM10 == $TUN} {set SIG14RA $SIG14R} else {set SIG14RC $SIG14R}
              }

    #
575 #   REM**Calculate SIG12RABC
    #SIG12R:
    #       SIG12RABC = REDREDRED
    #       IF OS4 = OCC THEN GOTO SIG8R
    #       IF SM8 = TUR THEN GOTO SM8REV
580 #       IF BK13 = OCC THEN GOTO SIG8R
    #       SIG12RABC = YELREDRED
    #       IF SIG14RA = RED THEN GOTO SIG8R
    #       SIG12RABC = GRNREDRED: GOTO SIG8R
    #SM8REV:
585 #       IF SM9 = TUR THEN SIG12RABC = REDREDYEL: GOTO SIG8R
    #       IF BK14 = OCC THEN GOTO SIG8R
    #       SIG12RABC = REDYELRED
```

```
590             set SIG12RABC $REDREDRED
                if {$OS4 != $OCC} {
                  if {$SM8 != $TUR} {
                    if {$BK13 != $OCC} {
                      set SIG12RABC $YELREDRED
595                   if {$SIG14RA != $RED} {set SIG12RABC $GRNREDRED}
                    }
                  } else {
                    # SM8REV:
                    if {$SM9 == $TUR} {
600                   set SIG12RABC $REDREDYEL
                    } elseif {$BK14 != $OCC} {
                      set SIG12RABC $REDYELRED
                    }
                  }
605             }


    #
    #  REM**Calculate SIG8RA and SIG8RC
610 #SIG8R:
    #       SIG8RA = RED: SIG8RC = RED
    #       IF DOT2 = WBD THEN GOTO SIG6R
    #       IF OS3 = OCC THEN GOTO SIG6R
    #       IF BK11 = OCC THEN GOTO SIG6R
615 #       SIG8R = YEL
    #       IF SIG12RABC = REDREDRED THEN GOTO SIG6R
    #       SIG8R = GRN
    #       IF SM7 = TUN THEN SIG8RA = SIG8R ELSE SIG8RC = SIG8R

620             set SIG8RA $RED; set SIG8RC $RED
                if {$DOT2 != $WBD && $OS3 != $OCC && $BK11 != $OCC} {
                  set SIG8R $YEL
                  if {$SIG12RABC != $REDREDRED} {set SIG8R $GRN}
                  if (SM7 == TUN) {set SIG8RA $SIG8R} else {set SIG8RC $SIG8R}
625             }


    #
    #  REM**Calculate SIG6RAB
630 #SIG6R:
    #       SIG6RAB = REDRED
    #       IF OS2 = OCC THEN GOTO SIG4R
    #       IF SM6 = TUR THEN GOTO SM6REV
```

```
    #       IF BK9 = OCC THEN GOTO SIG4R
635 #       SIG6RAB = YELRED
    #       IF SIG8RA = RED THEN GOTO SIG4R
    #       SIG6RAB = GRNRED: GOTO SIG4R
    #
    #SM6REV:
640 #       IF BK10 = OCC THEN GOTO SIG4R
    #       SIG6RAB = REDYEL

            set SIG6RAB $REDRED
            if {$OS2 != $OCC} {
645           if {$SM6 != $TUR} {
                if {$BK9 != $OCC} {
                  set SIG6RAB $YELRED
                  if {$SIG8RA != $RED} {set SIG6RAB $GRNRED}
                }
650           } else {
                # SM6REV:
                if {$BK10 != $OCC} {set SIG6RAB $REDYEL}
              }
            }
655
    #
    #  REM**Calculate SIG4RA
    #SIG4R:
    #       SIG4RA = RED
660 #       IF DOT1 = WBD THEN GOTO SIG2R
    #       IF BK8 = OCC THEN GOTO SIG2R
    #       SIG4RA = YEL
    #       IF SIG6RAB = REDRED THEN GOTO SIG2R
    #       SIG4RA = GRN
665
            set SIG4RA $RED
            if {$DOT1 != $WBD && $BK8 != $OCC} {
              set SIG4RA $YEL
              if {$SIG6RAB != $REDRED} {set SIG4RA $GRN}
670         }

    #
    #  REM**Calculate Red Butte exit signals
    #  REM**Initialize all exit signals to RED
675 #SIG2R:
    #       set SIG2RA $RED: SIG2RB = RED: SIG2RC = RED
    #       SIG2RD = RED: SIG2RE = RED: SIG2RF = RED
    #
```

```
#  REM**Calculate exit signal value for aligned track
#      EXITSIG = RED
#      IF DOT1 = WBD THEN GOTO SIGECMP
#      IF OS1 = OCC THEN GOTO SIGECMP
#      IF BK7 = OCC THEN GOTO SIGECMP
#      EXITSIG = YEL
#      IF SIG4RA = RED THEN GOTO EXSIG
#      EXITSIG = GRN
#
#  REM**Set track aligned exit signal to calculated signal value
#EXSIG:
#      IF TK = 1 THEN SIG2RA = EXITSIG
#      IF TK = 2 THEN SIG2RB = EXITSIG
#      IF TK = 3 THEN SIG2RC = EXITSIG
#      IF TK = 4 THEN SIG2RD = EXITSIG
#      IF TK = 5 THEN SIG2RE = EXITSIG
#      IF TK = 6 THEN SIG2RF = EXITSIG
#SIGECMP:

          set SIG2RA $RED; set SIG2RB $RED; set SIG2RC $RED
          set SIG2RD $RED; set SIG2RE $RED; set SIG2RF $RED
          set EXITSIG $RED
          if {$DOT1 != $WBD && $OS1 != $OCC && $BK7 != $OCC} {
            set EXITSIG $YEL
            if {$SIG4RA != $RED} {set EXITSIG $GRN}
          }
          switch $TK {
            1 {set SIG2RA $EXITSIG}
            2 {set SIG2RB $EXITSIG}
            3 {set SIG2RC $EXITSIG}
            4 {set SIG2RD $EXITSIG}
            5 {set SIG2RE $EXITSIG}
            6 {set SIG2RF $EXITSIG}
          }

#
#  REM*****************************
#  REM*****************************
#  REM**CALCULATE WESTBOUND SIGNALS**
#  REM*****************************
#  REM*****************************
#
#    REM**Calculate SIG2LAB (entrance signal into staging)
#SIG2L:
#      SIG2LAB = REDRED
```

```
    #       IF OS1 = OCC THEN GOTO SIG4L
725 #       IF TK = 1 AND BK1 = CLR THEN SIG2LAB = YELRED: GOTO SIG4L
    #       IF TK = 2 AND BK2 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
    #       IF TK = 3 AND BK3 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
    #       IF TK = 4 AND BK4 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
    #       IF TK = 5 AND BK5 = CLR THEN SIG2LAB = REDYEL: GOTO SIG4L
730 #       IF TK = 6 AND BK6 = CLR THEN SIG2LAB = REDYEL

              set SIG2LAB $REDRED
              if {$OS1 != $OCC} {
                if {$TK == $1 && $BK1 == $CLR} {
735               set SIG2LAB $YELRED
                } elseif {$TK == $2 && $BK2 == $CLR} {
                  set SIG2LAB $REDYEL
                } elseif {$TK == $3 && $BK3 == $CLR} {
                  set SIG2LAB $REDYEL
740             } elseif {$TK == $4 && $BK4 == $CLR} {
                  set SIG2LAB $REDYEL
                } elseif {$TK == $5 && $BK5 == $CLR} {
                  set SIG2LAB $REDYEL
                } elseif {$TK == $6 && $BK6 == $CLR} {
745               set SIG2LAB $REDYEL
                }
              }

    #
750 #   REM**Calculate SIG4LA
    #SIG4L:
    #       SIG4LA = RED
    #       IF DOT1 = EBD THEN GOTO SIG6L
    #       IF BK7 = OCC THEN GOTO SIG6L
755 #       SIG4LA = YEL
    #       IF SIG2LAB = REDRED THEN GOTO SIG6L
    #       SIG4LA = GRN

              set SIG4LA $RED
760           if {$DOT1 != $EBD && $BK7 != $OCC} {
                set SIG4LA $YEL
                if {$SIG2LAB != $REDRED} {set SIG4LA $GRN}
              }

765 #
    #   REM**Calculate SIG6LA and SIG6LC
    #SIG6L:
    #       SIG6LA = RED: SIG6LC = RED
```

```
    #       IF DOT1 = EBD THEN GOTO SIG8L
770 #       IF OS2 = OCC THEN GOTO SIG8L
    #       IF BK8 = OCC THEN GOTO SIG8L
    #       SIG6L = YEL
    #       IF SIG4LA = RED THEN GOTO SIG8L
    #       SIG6L = GRN
775 #       IF SM6 = TUN THEN SIG6LA = SIG6L ELSE SIG6LC = SIG6L
    #

            set SIG6LA $RED; set SIG6LC $RED
            if {$DOT1 != $EBD && $OS2 != $OCC && $BK8 != $OCC} {
780           set SIG6L $YEL
              if {$SIG4LA != $RED} {set SIG6L $GRN}
              if {$SM6 == $TUN} {set SIG6LA $SIG6L} else {set SIG6LC $SIG6L}
            }

785
    #  REM**Calculate SIG8LAB
    #SIG8L:
    #       SIG8LAB = REDRED
    #       IF OS3 = OCC THEN GOTO SIG12L
790 #       IF SM7 = TUR THEN GOTO SM7REV
    #       IF BK9 = OCC THEN GOTO SIG12L
    #       SIG8LAB = YELRED
    #       IF SIG6LA = RED THEN GOTO SIG12L
    #       SIG8LAB = GRNRED: GOTO SIG12L
795 #SM7REV:
    #       IF BK10 = OCC THEN GOTO SIG12L
    #       SIG8LAB = REDYEL

            set SIG8LAB $REDRED
800         if {$OS3 != $OCC} {
              if {$SM7 != $TUR} {
                if {$BK9 != $OCC} {
                  set SIG8LAB $YELRED
                  if {$SIG6LA != $RED} {set SIG8LAB $GRNRED}
805             }
              } else {
                # SM7REV:
                if {$BK10 != $OCC} {set SIG8LAB $REDYEL}
              }
810         }

    #
    #  REM**Calculate SIG12LA, SIG12LC and SIG12LD
```

```
    #SIG12L:
815 #       SIG12LA = RED: SIG12LC = RED: SIG12LD = RED
    #       IF DOT2 = EBD THEN GOTO SIG14L
    #       IF OS4 = OCC THEN GOTO SIG14L
    #       IF BK11 = OCC THEN GOTO SIG14L
    #       SIG12L = YEL
820 #       IF SIG8LAB = REDRED THEN GOTO SIG14L
    #       SIG12L = GRN
    #       IF SM8 = TUN THEN SIG12LA = SIG12L: GOTO SIG14L
    #       IF SM9 = TUN THEN SIG12LC = SIG12L ELSE SIG12LD = SIG12L

825           set SIG12LA $RED; set SIG12LC $RED; set SIG12LD $RED
              if {$DOT2 != $EBD && $OS4 != $OCC && $BK11 != $OCC} {
                set SIG12L $YEL
                if {$SIG8LAB != $REDRED} {set SIG12L $GRN}
                if {$SM8 == $TUN} {
830               set SIG12LA $SIG12L
                } elseif {$SM9 == $TUN} {
                  set SIG12LC $SIG12L
                } else {
                  set SIG12LD $SIG12L
835             }
              }


    #
840 #  REM**Calculate SIG14LAB
    #SIG14L:
    #       SIG14LAB = REDRED
    #       IF OS5 = OCC THEN GOTO SIG16L
    #       IF SM10 = TUR THEN GOTO SM10REV
845 #       IF BK13 = OCC THEN GOTO SIG16L
    #       SIG14LAB = YELRED
    #       IF SIG12LA = RED THEN GOTO SIG16L
    #       SIG14LAB = GRNRED: GOTO SIG16L
    #SM10REV:
850 #       IF BK14 = OCC THEN GOTO SIG16L
    #       SIG14LAB = REDYEL

              set SIG14LAB $REDRED
              if {$OS5 != $OCC} {
855             if {$SM10 != $TUR} {
                  if {$BK13 != $OCC} {
                    set SIG14LAB $YELRED
                    if {$SIG12LA != $RED} {set SIG14LAB $GRNRED}
```

```
                }
            } else {
                # SM10REV
                if {$BK14 != $OCC} {set SIG14LAB $REDYEL}
            }
        }

#
#  REM**Calculate SIG16LA
#SIG16L:
#       SIG16LA = RED
#       IF DOT3 = EBD THEN GOTO SIG18L
#       IF BK15 = OCC THEN GOTO SIG18L
#       SIG16LA = YEL
#       IF SIG14LAB = REDRED THEN GOTO SIG18L
#       SIG16LA = GRN

        set SIG16LA $RED
        if {$DOT3 != $EBD && $BK15 != $OCC} {
            set SIG16LA $YEL
            if {$SIG14LAB != $REDRED} {set SIG16LA $GRN}
        }

#
#  REM**Calculate SIG18LA
#SIG18L:
#       SIG18LA = RED
#       IF DOT3 = EBD THEN GOTO SIG20L
#       IF BK16 = OCC THEN GOTO SIG20L
#       SIG18LA = YEL
#       IF SIG16LA = RED THEN GOTO SIG20L
#       SIG18LA = GRN

        set SIG18LA $RED
        if {$DOT3 != $EBD && $BK16 != $OCC} {
            set SIG18LA $YEL
            if {$SIG16LA != $RED} {set SIG18LA $GRN}
        }

#
#  REM**Calculate SIG20LA, SIG20LB and SIG20LD
#SIG20L:
#       SIG20LA = RED: SIG20LB = RED: SIG20LD = RED
#       IF DOT3 = EBD THEN GOTO SIGWCMP
#       IF OS6 = OCC THEN GOTO SIGWCMP
```

```
    #       IF BK17 = OCC THEN GOTO SIGWCMP
905 #       SIG20L = YEL
    #       IF SIG18LA <> RED THEN SIG20L = GRN
    #       IF SM11 = TUR THEN SIG20LB = SIG20L: GOTO SIGWCMP
    #       IF TF12 = TGR THEN SIG20LD = SIG20L ELSE SIG20LA = SIG20L
    #SIGWCMP:
910
            set SIG20LA $RED; set SIG20LB $RED; set SIG20LD $RED
            if {$DOT3 != $EBD && $OS6 != $OCC && $BK17 != $OCC} {
              set SIG20L $YEL
              if {$SIG18LA != $RED} {set SIG20L $GRN}
915           if {$SM11 == $TUR} {
                set SIG20LB $SIG20L
              } elseif {$TF12 == $TGR} {
                set SIG20LD $SIG20L
              } else {
920             set SIG20LA $SIG20L
              }
            }
    #
    #   REM**PACK AND WRITE OUTPUT BYTES FOR NODE 0 (SMINI)
925 #       OB(1) = SIG2LAB                 'NODE 0 CARD 0 PORT A
    #       OB(1) = SIG2RA * B5 OR OB(1)

            set Outputs [expr {$SIG2LAB | $SIG2RA << $B5}];
    #NODE 0 CARD 0 PORT A

930 #
    #       OB(2) = SIG2RB                  'NODE 0 CARD 0 PORT B
    #       OB(2) = SIG2RC * B3 OR OB(2)
    #       OB(2) = SM1 * B6 OR OB(2)

935         lappend Outputs [expr {$SIG2RB | \
                                   $SIG2RC << $B3 | \
                                   $SM1    << $B6}];          #NODE 0 CARD 0 PORT B
    #
    #       OB(3) = SIG2RD                  'NODE 0 CARD 0 PORT C
940 #       OB(3) = SIG2RE * B3 OR OB(3)
    #       OB(3) = SM2 * B6 OR OB(3)

            lappend Outputs [expr {$SIG2RD | \
                                   $SIG2RE << $B3 | \
945                                $SM2    << $B6}];          #NODE 0 CARD 0 PORT C
    #
    #       OB(4) = SIG2RF                  'NODE 0 CARD 1 PORT A
```

```
    #       OB(4) = SM3 * B3 OR OB(4)
    #       OB(4) = SM4 * B5 OR OB(4)
950
              lappend Outputs [expr {$SIG2RF | \
                                     $SM3    << $B3 | \
                                     $SM4    << $B5}];          #NODE 0 CARD 1 PORT A
    #
955 #       OB(5) = SM5                         'NODE 0 CARD 1 PORT B

              lappend Outputs $SM5;#
    NODE 0 CARD 1 PORT B

    #       UA = 0: NO = 6
960 #       CALL OUTPUTS

              set UA 0
              if {[catch {bus Outputs $Outputs $UA} result]} {
                      # Handle error.
965                   puts -nonewline stderr "Could not write to the output ports of "
                      puts stderr "SMINI card at UA $UA: $result"
                      break
              }

970


    #
    #  REM**PACK AND WRITE OUTPUT BYTES FOR NODE 1 (SMINI)
    #       OB(1) = SIG4LA                      'NODE 1 CARD 0 PORT A
975 #       OB(1) = SIG4RA *B3 OR OB(1)
    #       OB(1) = SM6 * B6 OR OB(1)

              set Outputs [expr {$SIG4LA | \
                                 $SIG4RA << $B3 | \
980                              $SM6    << $B6}];     #NODE 1 CARD 0 PORT A

    #
    #       OB(2) = SIG6RAB                     'NODE 1 CARD 0 PORT B
    #       OB(2) = SIG6LA * B5 OR OB(2)
985
              lappend Outputs [expr {$SIG6RAB | \
                                     $SIG6LA << $B5}];          #NODE 1 CARD 0 PORT B
    #
    #       OB(3) = SIG6LC                      'NODE 1 CARD 0 PORT C
990 #       OB(3) = SIG8RA * B3 OR OB(3)
    #       OB(3) = SM7 * B6 OR OB(3)
```

```
            lappend Outputs [expr {$SIG6LC | \
                                   $SIG8RA << $B3 | \
995                                $SM7    << $B6}];          #NODE 1 CARD 0 PORT B
    #
    #       OB(4) = SIG8LAB                  'NODE 1 CARD 1 PORT A
    #       OB(4) = SIG8RC * B5 OR OB(4)

1000        lappend Outputs [expr {$SIG8LAB | \
                                   $SIG8RC  << $B5}];         #NODE 1 CARD 1 PORT A
    #       UA = 1: NO = 6
    #       CALL OUTPUTS

1005        set UA 1
            if {[catch {bus Outputs $Outputs $UA} result]} {
                    # Handle error.
                    puts -nonewline stderr "Could not write to the output ports of "
                    puts stderr "SMINI card at UA $UA: $result"
1010                break
            }



1015 #
    #   REM**PACK AND WRITE OUTPUT BYTES FOR NODE 2 (SMINI)
    #       OB(1) = SIG12RABC                'NODE 2 CARD 0 PORT A

            set Outputs $SIG12RABC;          #NODE 2 CARD 0 PORT A
1020
    #
    #       OB(2) = SIG12LA                  'NODE 2 CARD 0 PORT B
    #       OB(2) = SIG12LC * B3 OR OB(2)
    #       OB(2) = SM8 * B6 OR OB(2)
1025
            lappend Outputs [expr {$SIG12LA | \
                                   $SIG12LC << $B3 | \
                                   $SM8    << $B6}];          #NODE 2 CARD 0 PORT B
    #
1030 #       OB(3) = SIG12LD                  'NODE 2 CARD 0 PORT C
    #       OB(3) = SIG14RA * B3 OR OB(3)
    #       OB(3) = SM9 * B6 OR OB(3)

            lappend Outputs [expr {$SIG12LD | \
1035                               $SIG14RA << $B3 | \
                                   $SM9    << $B6}];          #NODE 2 CARD 0 PORT C
```

```
    #
    #       OB(4) = SIG14LAB                    'NODE 2 CARD 1 PORT A
    #       OB(4) = SIG14RC * B5 OR OB(4)
1040
            lappend Outputs [expr {$SIG14LAB | \
                                   $SIG14RC << $B5}];          #NODE 2 CARD 1 PORT A
    #
    #       OB(5) = SIG16LA                     'NODE 2 CARD 1 PORT B
1045 #       OB(5) = SIG16RA * B3 OR OB(5)
    #       OB(5) = SM10 * B6 OR OB(5)

            lappend Outputs [expr {$SIG16LA | \
                                   $SIG16RA << $B3 | \
1050                               $SM10    << $B6}];          #NODE 2 CARD 1 PORT B
    #       UA = 2: NO = 6
    #       CALL OUTPUTS

            set UA 2
1055        if {[catch {bus Outputs $Outputs $UA} result]} {
                    # Handle error.
                    puts -nonewline stderr "Could not write to the output ports of "
                    puts stderr "SMINI card at UA $UA: $result"
                    break
1060        }

    #
    #   REM**PACK AND WRITE OUTPUT BYTES FOR NODE 3 (SMINI)
    #       OB(1) = SIG18LA                     'NODE 3 CARD 0 PORT A
1065 #       OB(1) = SIG18RA * B3 OR OB(1)
    #       OB(1) = SM11 * B6 OR OB(1)

            set Outputs [expr {$SIG18LA | \
                               $SIG18RA << $B3 | \
1070                           $SM11    << $B6}];    #NODE 3 CARD 0 PORT A

    #
    #       OB(2) = SIG20RABC                   'NODE 3 CARD 0 PORT B

1075        lappend Outputs $SIG20RABC;       #NODE 3 CARD 0 PORT B
    #
    #       OB(3) = SIG20LA
    #       OB(3) = SIG20LB * B3 OR OB(3)     'NODE 3 CARD 0 PORT C

1080        lappend Outputs [expr {$SIG20LA | \
                                   $SIG20LB << $B3}];          #NODE 3 CARD 0 PORT C
```

```
#
#       OB(4) = SIG20LD                         'NODE 3 CARD 1 PORT A

1085            lappend Outputs $SIG20LD;         #NODE 3 CARD 1 PORT A
#       UA = 3: NO = 6
#       CALL OUTPUTS

                set UA 3
1090            if {[catch {bus Outputs $Outputs $UA} result]} {
                        # Handle error.
                        puts -nonewline stderr "Could not write to the output ports of "
                        puts stderr "SMINI card at UA $UA: $result"
                        break
1095            }
#
#   REM**PACK AND WRITE OUTPUT BYTES FOR NODE 4 (SUSIC)
#       OB(1) = DOT1                            'NODE 4 CARD 0 PORT A
#       OB(1) = DOT2 * B2 OR OB(1)
1100 #      OB(1) = DOT3 * B4 OR OB(1)
#       OB(1) = BK1 * B6 OR OB(1)
#       OB(1) = BK2 * B7 OR OB(1)

                set Outputs [expr {$DOT1 | \
1105                                $DOT2 << $B2 | \
                                    $DOT3 << $B4 | \
                                    $BK1  << $B6 | \
                                    $BK2  << $B7}];       #NODE 4 CARD 0 PORT A

1110 #
#       OB(2) = BK3 * B0 OR OB(2)           'NODE 4 CARD 0 PORT B
#       OB(2) = BK4 * B1 OR OB(2)
#       OB(2) = BK5 * B2 OR OB(2)
#       OB(2) = BK6 * B3 OR OB(2)
1115 #      OB(2) = OS1 * B4 OR OB(2)
#       OB(2) = BK7 * B5 OR OB(2)
#       OB(2) = BK8 * B6 OR OB(2)
#       OB(2) = OS2 * B7 OR OB(2)

1120            lappend Outputs [expr {$BK3 | \
                                       $BK4 << $B1 | \
                                       $BK5 << $B2 | \
                                       $BK6 << $B3 | \
                                       $OS1 << $B4 | \
1125                                   $BK7 << $B5 | \
                                       $BK8 << $B6 | \
```

```
                                    $OS2 << $B7}];    #NODE 4 CARD 0 PORT B
    #
    #     OB(3) = BK9 * B0 OR OB(3)        'NODE 4 CARD 0 PORT C
    #     OB(3) = BK10 * B1 OR OB(3)
    #     OB(3) = OS3 * B2 OR OB(3)
    #     OB(3) = BK11 * B3 OR OB(3)
    #     OB(3) = OS4 * B4 OR OB(3)
    #     OB(3) = BK13 * B5 OR OB(3)
    #     OB(3) = BK14 * B6 OR OB(3)
    #     OB(4) = OS5 * B7 OR OB(4)

          lappend Outputs [expr {$BK9  | \
                                    $BK10 << $B1 | \
                                    $OS3  << $B2 | \
                                    $BK11 << $B3 | \
                                    $OS4  << $B4 | \
                                    $BK13 << $B5 | \
                                    $BK14 << $B6 | \
                                    $OS5  << $B7}];  #NODE 4 CARD 0 PORT C
    #
    #     OB(4) = BK15 * B0 OR OB(4)       'NODE 4 CARD 0 PORT D
    #     OB(4) = BK16 * B1 OR OB(4)
    #     OB(4) = BK17 * B2 OR OB(4)
    #     OB(4) = OS6 * B3 OR OB(4)
    #     OB(4) = BK18 * B4 OR OB(4)

          lappend Outputs [expr {$BK15 | \
                                    $BK16 << $B1 | \
                                    $BK17 << $B2 | \
                                    $OS6  << $B3 | \
                                    $BK18 << $B4}];  #NODE 4 CARD 0 PORT D
    #     UA = 4: NO = 4
    #     CALL OUTPUTS

          set UA 4
          if {[catch {bus Outputs $Outputs $UA} result]} {
                # Handle error.
                puts -nonewline stderr "Could not write to the output ports of "
                puts stderr "SUISC card at UA $UA: $result"
                break
          }
    #
    #  REM**RETURN TO BEGINNING OF REAL-TIME LOOP
    #     GOTO BRTL
```

```
}
```

Chapter 6

XPressNet Programming

Chapter 7

Creating Splash Windows

Splash windows are windows that are displayed during program startup and generally feature a company or program logo, some introductory text and a progress meter and status text area describing the program startup and initialization progress.

7.1 Creating a Splash Window

Listing 7.1: Snit spash widget creation

```
spash widgetpath [options...]
```

The splash widget is implemented as a Snit widget type and are created with the spash procedure. This widget takes eight options:

-troughcolor The color to use for the progressbar's trough.

-titleforeground The foreground color for the title text area.

-statusforeground The foreground color for the status text area.

-background The overall background color.

-progressbar Whether or not to include a progress bar.

-image The image to display as the main feature of the spash screen (typically a company logo or product image).

-icon A small image to display next to the title text.

-title Text to display above the main spash image.

The splash widget defines four public methods:

update This method takes two arguments, statusMessage percentDone, which supply the text for the status message and the percent of the startup that has completed. If the completion done percentage is 100, the spash screen's "click to destroy" function is enabled.

enableClickDestroy This method enables the spash screen's "click to destroy" function.

hide This method hides (withdraws) the splash window.

show This method shows (unwithdraws) the splash window.

7.2 Typical usage

Listing 7.2: Typical usage

```
image create photo DeepwoodsBanner -format gif \
        -file [file join $CommonImageDir DeepwoodsBanner.gif]
# Deepwoods banner image.  Used in the splash screen.
# [index] DeepwoodsBanner!image

proc SplashScreen {} {
  # Build the ``Splash Screen'' -- A popup window that
  # tells the user what we are all about.  It gives the
  # version and brief copyright information.
  #
  # The upper part of the splash screen gives the brief
  # information, with directions on how to get detailed
  # information.  The lower part contains an image banner
  # for Deepwoods Software.
  # [index] SplashScreen!procedure

  splash .mrrSplash \
        -title {Sample Code Program -- sample code for
Programming Guide, Copyright (C) 2005 Robert Heller D/B/A
Deepwoods Software Model Railroad Timetable Chart Program
comes with ABSOLUTELY NO WARRANTY; for details select
'Warranty...' under the Help menu.  This is free software,
and you are welcome to redistribute it under certain
conditions; select 'Copying...' under the Help menu.} \
```

```
25          -image DeepwoodsBanner -background {#2ba2bf} \
            -titleforeground white -statusforeground {black}
    }

    proc SplashWorkMessage {message percent} {
30    .mrrSplash update "$message" $percent
      update idle
    }

    wm withdraw .;# Withdraw the main window.
35  SplashScreen;# Create the splash screen.
    update idle;# Flush pending idle events.

    catch {SplashWorkMessage {Creating Main Window} 11}

40  # Create the main window...

    catch {SplashWorkMessage {Create CTC Panel} 22}

    # Create the CTC Panel...
45
    catch {SplashWorkMessage {Create Configutation} 33}

    # Create the Configutation...

50  catch {SplashWorkMessage {Done} 100}
```

Listing 7.2 shows how a splash window is typically created and updated. The procedure in lines 6-28 creates the splash window and the procedure in lines 29-32 updates the status text area and progress bar. After withdrawing the main window, the splash window is created and drawn (lines 34-36). As progress is made during program startup, the splash window is updates (lines 38, 42, 46, and 50). When the percent completed is set to 100, the splash window can be removed when the user left-clicks anywhere on it. The splash window this code creates is shown in Figure 7.1.

Figure 7.1: Sample Splash Screen

Chapter 8

Creating Main Windows

To aid in creating standardized main windows with some common "trimmings", the Model Railroad System includes a package that enhances and extends the BWidget MainWindow widget. The enhanced main window widget is a Snit widgetadaptor that adds a BWidget ScrolledWindow, an enhanced PanedWindow, a ButtonBox, and "slideouts"[1] to the body of the BWidget MainWindow. Along with the enhanced main window, there is also code available for a standardized menu bar[2].

8.1 What is in an enhanced Main Window?

An enhanced Main Window is a BWidget MainWindow, with an enhanced PanedWindow, with a ScrolledWindow on the left and a vertical ButtonBox on the right, with zero or more optional "slideouts" to the right of the vertical ButtonBox. The ScrolledWindow can contain any scrollable widget, such as a text window (for log output), a canvas (for graphical output), a ListBox or Tree widget, or a ScrolledFrame. The vertical ButtonBox can be used for a vertical collection of command buttons.

8.1.1 Slideouts

Slideouts are additional frames that can be "slid" out (exposed to the right) as needed and then slid back in (withdraw from view) when no longer needed. Useful

[1] See Section 8.1.1 for more information about slideouts.
[2] See Section 8.1.2 for more information about the standardized menu bar.

for GUI elements that don't need to be displayed all of the time. Any number of these slideouts can be created and any subset of them can be displayed at any time.

8.1.2 Standard Menu Bars

The BWStdMenuBar package defines code for a standardized menu bar containing a File, Edit, View, Options, and Help drop down menus. The File, Edit, and Help menus are populated with standard menu items.

8.2 Creating a typical Main Window

Listing 8.1: Creating a Main Window

```
#*
#*

package require Tk
package require snit
package require snitStdMenuBar;#          Standard Menu Bar
package require HTMLHelp 2.0;#   Help package
package require MainWindow;#     Main Window package

namespace eval SampleCode {}

proc SampleCode::SampleCodeMain {} {

  wm protocol . WM_DELETE_WINDOW {SampleCode::CarefulExit}
  wm title . "Sample Code"

  variable Main;#                 Main window
  variable MainWindow;#           Main display area
  variable CanvasWindow;#         Canvas window

  # Create the main window
  pack [set Main [mainwindow .main -dontwithdraw no]] \
        -expand yes -fill both
  # Create and show a toolbar --
  #     this toolbar will mirror the File menu
  $Main toolbar add tools
  $Main toolbar show tools
  # Disable the unused File menu items
  $Main menu entryconfigure file New -state disabled
```

```
    $Main menu entryconfigure file Open... -state disabled
    $Main menu entryconfigure file Save -state disabled
    $Main menu entryconfigure file {Save As...} -state disabled
    $Main menu entryconfigure file Print... -state disabled
34  # Bind Close and Exit menu items to the exit function
    # and create an exit button.
    $Main menu entryconfigure file Close -command SampleCode::CarefulExit
    $Main menu entryconfigure file Exit -command SampleCode::CarefulExit
    $Main toolbar addbutton tools close -image CloseButtonImage \
39                                  -command SampleCode::CarefulExit
    set MainWindow [$Main scrollwindow getframe]
    # Create a canvas window
    set CanvasWindow [canvas $MainWindow.canvas -background white]
    $Main scrollwindow setwidget $CanvasWindow
44  # Add a 'slideout'
    variable Slideout [$Main slideout add sampleslide]
    # Add some buttons to the Button box
    $Main buttons add ttk::button showslide \
                      -text "Show Slideout" \
49                    -command "$Main slideout show sampleslide"
    $Main buttons add ttk::button hideslide \
                      -text "Hide Slideout" \
                      -command "$Main slideout hide sampleslide"
    $Main buttons add ttk::button testB1 \
54                    -text "Test Button 1" \
                      -command {tk_messageBox -type ok \
                                              -icon info \
                                              -message "Testing"}

59  $Main menu delete help "On Keys..."
    $Main menu delete help "Index..."
    $Main menu add help command \
          -label "Reference Manual" \
          -command "HTMLHelp help {Sample Reference}"
64  $Main menu entryconfigure help "On Help..." \
          -command "HTMLHelp help Help"
    $Main menu entryconfigure help "Tutorial..." \
          -command "HTMLHelp help {Sample Tutorial}"
    $Main menu entryconfigure help "On Version" \
69        -command "HTMLHelp help Version"
    $Main menu entryconfigure help "Copying" \
          -command "HTMLHelp help Copying"
    $Main menu entryconfigure help "Warranty" \
          -command "HTMLHelp help Warranty"
74
```

```
    HTMLHelp setDefaults "$::HelpDir" "index.html#toc"

79  }

    proc SampleCode::CarefulExit {{ask 1}} {
      if {$ask} {
        set ans [tk_messageBox -type yesno -icon question -message {Really Exit?}]
84    } else {
        set ans 1
      }
      if {!$ans} {return}
      exit
89  }

    package provide SampleCodeMain 1.0
```

Figure 8.1: Sample Main Window, slideout hidden

Listing 8.1 contains code to create a basic Main Window with a canvas window,

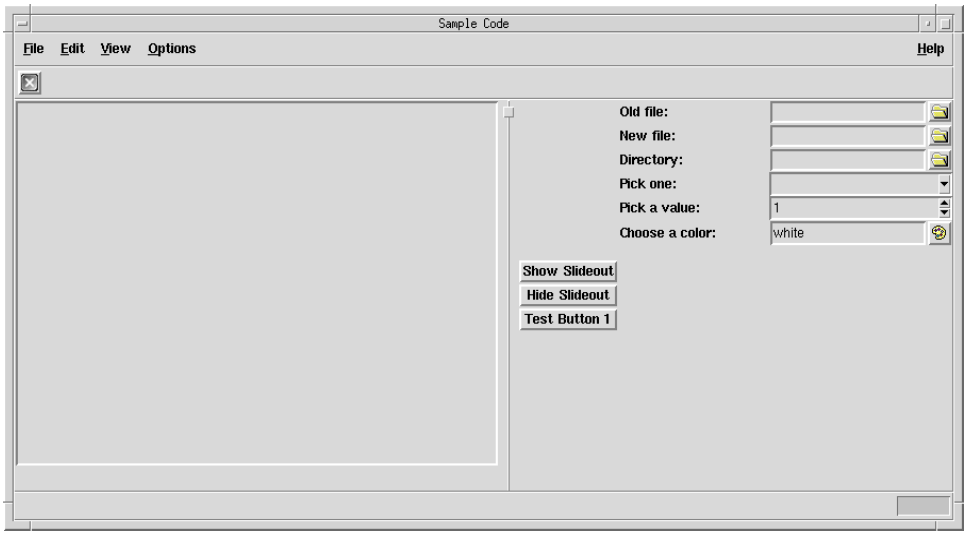

Figure 8.2: Sample Main Window, slideout shown

simple set of command buttons, and a slideout[3]. The window this code creates is shown in Figures 8.1 and 8.2.

[3]See Chapter 9 for the code used to populate the slideout.

Chapter 9

An Assortment of LabelFrame Based Widgets.

Included with the Model Railroad system are four packages that define a a collection of BWidget LabelFrame based widgets. These widgets are much line the BWidget LabelEntry widget, except that instead of a plain Entry widget, they have other, more specialized widgets.

The four packages are:

1. BWFileEntry defines the FileEntry widget, which is meant to enter file and directory names.
2. BWLabelComboBox defines the LabelComboBox widget, which has a ComboBox widget instead of an Entry widget.
3. BWLabelSpinBox defines the LabelSpinBox widget, which has a SpinBox widget instead of an Entry widget.
4. LabelSelectColor defines the LabelSelectColor widget, which is meant to select a color.

Listing 9.1: Using the LabelFrame based widgets

```
# $Id: SampleCodeSlideout.tcl 1709 2014-05-21 16:02:04Z heller $

package require LabelFrames;# Label<mumble> (LabelEntry, LabelSpinBox, LabelComboBox)
5
# Populate the sample slideout with various LabelEntry like
```

```
# Widgets.  These widgets all have a Label and some kind of
# Entry-like widget for gathering some sort of user input.

namespace eval SampleCode {}
proc SampleCode::SampleCodeSlideout {} {
  variable Slideout

  # FileEntry widgets are used to get file and directory
  # names.  There are three types: old (existing) files,
  # generally for use as input files, new files for saving
  # information to, and directories (folders).
  #
  # First an old file:
  pack [FileEntry $Slideout.oldfile \
                -label "Old file:" -labelwidth 20\
                -filedialog open -defaultextension .text \
                -filetypes {
                        {{Text Files} {.text .txt} TEXT}
                        {{All Files}  *            }}] \
        -fill x
  # Second a new file:
  pack [FileEntry $Slideout.newfile \
                -label "New file:" -labelwidth 20\
                -filedialog save -defaultextension .text \
                -filetypes {
                        {{Text Files} {.text .txt} TEXT}
                        {{All Files}  *            }}] \
        -fill x
  # Finally, a directory:
  pack [FileEntry $Slideout.directory \
                -label "Directory:" -labelwidth 20\
                -filedialog directory] -fill x
  # A LabelComboBox would be used for selecting
  # from a list of values.
  pack [LabelComboBox $Slideout.combo \
                -label "Pick one:" -labelwidth 20\
                -values {A B C D}] -fill x
  # A LabelSpinBox would be usually be used for
  # selecting a number from a range.
  pack [LabelSpinBox $Slideout.spin \
                -label "Pick a value:" -labelwidth 20\
                -range {1 10 1}] -fill x
  # A LabelSelectColor would be used to select a
  # color.
  pack [LabelSelectColor $Slideout.color \
```

```
                -label "Choose a color:" -labelwidth 20] \
        -fill x
}

package provide SampleCodeSlideout 1.0
```

Listing 9.1 contains code to create each of the LabelFrame widgets. The window this code helped to create is shown in Figure 8.2

Chapter 10

HTMLHelp

10.1 About

The HTMLHelp widget is a help dialog that display program documentation coded in a simple subset of HTML. There is limited support for Cascading Style Sheets. There is a support for searching and there is a simple history stack.

10.2 Using the HTMLHelp widget

The HTMLHelp is a SNIT widgetadaptor based on the BWidget Dialog widget. It takes these options:

-textwidth The width of the help text component (see -width of the text widget).

-width The width of the HTMLHelp (see -width of the Dialog widget).

-height The height of the HTMLHelp (see -height of the Dialog widget).

-side The side to place the pane slider button. Can be top (the default) or bottom (see -side of the PanedWindow widget). This is a read only option.

-helpdirectory The name of the directory where the HTML code lives. This is a read only option. There is no default value and this option must be specified. (But see below for the typemethod setDefaults.)

-tableofcontents The name of the HTML file in the help directory that contains the Table Of Contents. This is a read only option. There is no default

value and this option must be specified. (But see below for the typemethod setDefaults.)

There is one public method available:

```
$HTMLHelpObject helptopic topicstring
```

This method searches the table of contents for a hyperlink with the specified help topic string as the link text (case folded search) and opens the HTMLHelp dialog with the page specified by the link associated with this text.

There are two public typemethods available:

```
HTMLHelp::HTMLHelp setDefaults helpdir tocfile
```

This method sets the default values for the `-helpdirectory` and `-tableofcontents` options.

```
HTMLHelp::HTMLHelp help topicstring
```

This typemethod just calls the `helptopic` method for a default help dialog object. If a default help does not exist, it is created, using the default values set by the `setDefaults` typemethod.

10.3 Creating Help text with LaTeX and tex4ht

Help text HTML files can be generated using LaTeX and tex4ht. You need be sure you include a table of contents (`\tableofcontents`) and use the htlatex command with `html,4,info` as its second parameter. The former creates the index for the left column and the latter breaks the file up into one file for each section. Be sure to include a chapter or section titled "Help"[1], since this is used as the topic text for the HTMLHelp dialog itself. Generally you would write a chapter, section or subsection for each help topic you expect to have for your application – that is for all of the Help buttons on dialogs and Help menu items on main windows[2].

[1] A sample chapter is included with the development code as Help.tex.

[2] See SampleCode.tex for a sample documentation LaTeX source file.

Chapter 11

Creating CTC Panels

Version 2 of the CTC Panel code is covered in this chapter. Version 1 is depreciated and is included only for older applications.

11.1 Creating a CTCPanel and creating objects to populate it.

Listing 11.1: CTCPanel::CTCPanel procedure

```
CTCPanel::CTCPanel widgetpath [options...]
```

CTCPanels are implemented with a Snit widget type and are created with the CTCPanel::CTCPanel procedure, as shown in Listing 11.1. This widget takes four options:

-schematicbackground The background color of the schematic display. Defaults to black.

-controlbackground The background color of the control display. Defaults to darkgreen.

-width The total width of the megawidget.

-height The total height of the megawidget.

Listing 11.2: CTCPanel::CTCPanel creating objects

```
$CTCPanelObject create type name [options...]
```

Method	Description
getv	Gets the current value of the object.
setv	Set the value of the object.
geti	Gets the value of the specified indicator.
seti	Sets the value of the specified indicator.
itemcget	Gets the value of a specified option of the object.
itemconfigure	Sets the value of a specified option of the object.
exists	Tests to see of the object exists.
delete	Deletes the specified object.
move	Moves the specified object a relative distance.
class	Returns the class (or type) of the object.
invoke	Executes a script bound to an object.
coords	Returns the coordinates of a specified terminal element of the object.
print	Writes a scriptlet to re-create the object to the specified output stream.

Table 11.1: Object methods

The panel includes its own scrollbars, including a shared horizontal scrollbar. Items displayed on the CTCPanel (either as schematic trackwork in the schematic display or as control items in the control display) are created with the create command, as shown in Listing 11.2. Each object has a unique name, a value, zero or more indicators, and zero or more options. The CTCPanel widget defines thirteen methods that take an object name, as shown in Table 11.1.

11.2 Control Points

Objects on a CTCPanel are grouped into named Control Points. A Control Point defines one (or more) trackwork elements and their corresponding control panel elements. In the simplest case a control point might include a Switch (turnout), a SIGPlate, a SWPlate, and a CodeButton, with the SWPlate controlling the Switch and the SIGPlate controlling the signals surrounding the Switch.

11.3 Object values and invocation

Many objects have an associated value. Generally, these values represent state information and these values (or states) correspond to different scripts bound to the object. For example trackwork switches (turnouts) define their state as being normal (points aligned for the main line) or reversed (points aligned for the branch line).

Also many objects have **–command** options, which define scriptlets to invoke under certain conditions. One (or more) of these scriptlets are executed (at the global level) when the object is invoked. Generally, trackwork objects are invoked to acquire state information (such as occupancy or point state feedback) from input ports bits (such as read from a Chubb CMR/I serial bus), and control objects are invoked to read and effect control panel settings by setting output port bits for trackwork control and signals. Generally, the **–command** bound to a code button will in turn invoke all of the objects associated with the control point the code button manages.

All trackwork objects have an –occupiedcommand option, which is a script to run to fetch the object's occupied state. The invoke method always returns the occupied state (always false for objects that are not trackwork objects). Trackwork with movable points should not have their points moved which a train is occupying the trackwork! Generally, if any of the trackwork of a control point is occupied,

the control point's code button should not invoke any of the control elements.

11.4 Sample CTCPanel and the code to create it.

Listing 11.3: Creating a CTC Panel

```
#*
#*

package require CTCPanel 2.0;#  Include CTCPanel V2.0

namespace eval SampleCode::CTCPanel {}
  # Procedure bound to the switch plate commands.
  proc SampleCode::CTCPanel::SetSwitchSP1 {state} {
    variable CTCPanel
    # Just move the switch's points.
    $CTCPanel setv IndustrySwitch $state
    # And set the indicators.
    switch $state {
      Normal {
        $CTCPanel seti SP1Switch N on
        $CTCPanel seti SP1Switch R off
      }
      Reverse {
        $CTCPanel seti SP1Switch R on
        $CTCPanel seti SP1Switch N off
      }
    }
  }
  # Procedure bound to the signal plate commands
  proc SampleCode::CTCPanel::SetSignalSP1 {state} {
    variable CTCPanel
    # Set the signal indicator lamps.
    switch $state {
      Left {
        $CTCPanel seti SP1Signal L on
        $CTCPanel seti SP1Signal R off
        $CTCPanel seti SP1Signal C off
      }
      Right {
        $CTCPanel seti SP1Signal L off
        $CTCPanel seti SP1Signal R on
        $CTCPanel seti SP1Signal C off
      }
```

```
      Center {
        $CTCPanel seti SP1Signal L off
        $CTCPanel seti SP1Signal R off
        $CTCPanel seti SP1Signal C on
      }
    }
  }
  # Procedure bound to the code button
  proc SampleCode::CTCPanel::CodeSP1 {} {
    variable CTCPanel
    # Check if the switch is occupied
    # (and fetch any state feedback).
    if {[$CTCPanel invoke IndustrySwitch]} {
      # Switch is occupied: controls are disabled.
      $CTCPanel setv SP1Signal Center
      $CTCPanel invoke SP1Signal
      return false
    } else {
      # If the switch is not occupied, invoke
      # the switch and signal plates.
      $CTCPanel invoke SP1Switch
      $CTCPanel invoke SP1Signal
      return true
    }
  }
  # procedure to show the CTC Panel.
  proc SampleCode::CTCPanel::ShowCTCPanel {} {
    variable CTCPanelTL
    wm deiconify $CTCPanelTL
  }

proc SampleCode::CTCPanel::SampleCodeCTCPanel {} {

  $::SampleCode::Main menu add view command \
                -label CTCPanel \
                -command ::SampleCode::CTCPanel::ShowCTCPanel

  # Create the CTC Panel on a new, transient toplevel.
  # Toplevel where Sample CTCPanel lives
  variable CTCPanelTL ${::SampleCode::Main}.ctcPanelTop
  toplevel $CTCPanelTL
  wm withdraw $CTCPanelTL
  wm transient $CTCPanelTL [winfo toplevel ${::SampleCode::Main}]
  wm title $CTCPanelTL "Sample CTC Panel"
  wm protocol $CTCPanelTL WM_DELETE_WINDOW "wm withdraw $CTCPanelTL"
```

```
    # Create a main window
    set panelMain [MainFrame ${CTCPanelTL}.main]
    # With a toolbar
    set panelToolbar [$panelMain addtoolbar]
    # Close button
    pack [ttk::button $panelToolbar.close \
                -image CloseButtonImage \
                -command "wm withdraw $CTCPanelTL"] \
          -side right
    pack $panelMain -fill both -expand yes
    # Create a CTC Panel
    variable CTCPanel [::CTCPanel::CTCPanel \
                          [$panelMain getframe].panel \
                          -width 400]
    pack $CTCPanel -fill both -expand yes

    # Populate the CTC Panel:

    # Control Point Block14:
    # Just a block of the main line to the west.
    $CTCPanel create StraightBlock MainWest \
                -x1 10 -x2 150 -y1 50 -y2 50 \
                -controlpoint Block14 \
                -label "Block 14" -position above

    # Control Point SP1:
    # The Switch itself.
    $CTCPanel create Switch IndustrySwitch \
                -x 150 -y 50 -controlpoint SP1 \
                -label "SP1"

    # The Switch Plate
    $CTCPanel create SWPlate SP1Switch \
          -x 150 -y 75 -controlpoint SP1 \
          -label "SP1" \
          -normalcommand \
          "::SampleCode::CTCPanel::SetSwitchSP1 Normal" \
          -reversecommand \
          "::SampleCode::CTCPanel::SetSwitchSP1 Reverse"

    # The signal plate
    $CTCPanel create SIGPlate SP1Signal -x 150 -y 150 \
          -controlpoint SP1 \
          -label "SP1" \
          -leftcommand \
```

```
            "::SampleCode::CTCPanel::SetSignalSP1 Left" \
130         -rightcommand \
            "::SampleCode::CTCPanel::SetSignalSP1 Right" \
            -centercommand \
            "::SampleCode::CTCPanel::SetSignalSP1 Center"

135   # The code button
      $CTCPanel create CodeButton SP1Code -x 250 -y 150 \
            -controlpoint SP1 \
            -command "::SampleCode::CTCPanel::CodeSP1"

140   # A label for the control point
      $CTCPanel create CTCLabel SP1Label -x 200 -y 200 \
            -controlpoint SP1 -label "SP1"

      # Initialize the control point
145   ::SampleCode::CTCPanel::CodeSP1

      # Control Point Block15:
      # Just a block of the main line east.
      set me1 [$CTCPanel coords IndustrySwitch Main]
150   set mex1 [lindex $me1 0]
      set mey  [lindex $me1 1]
      set mex2 [expr {$mex1 + 150}]
      $CTCPanel create StraightBlock MainEast \
                  -x1 $mex1 -x2 $mex2 -y1 $mey -y2 $mey -controlpoint Block15 \
155               -label "Block 15"  -position above


      # Control Point Spur1:
      # An industrial spur.
160   set is1 [$CTCPanel coords IndustrySwitch Divergence]
      set isx1 [lindex $is1 0]
      set isy  [lindex $is1 1]
      set isx2 [expr {$isx1 + 150}]
      $CTCPanel create StraightBlock IndustrySpur \
165               -x1 $isx1 -x2 $isx2 -y1 $isy -y2 $isy \
                  -controlpoint Spur1 -label "Spur 1"

}

170 package provide SampleCodeCTCPanel 1.0
```

The Tcl code in Listing 11.3 creates a simple CTC Panel[1]. The CTC Panel itself

[1] See [4], Section 1.4 for a complete list of available panel elements.

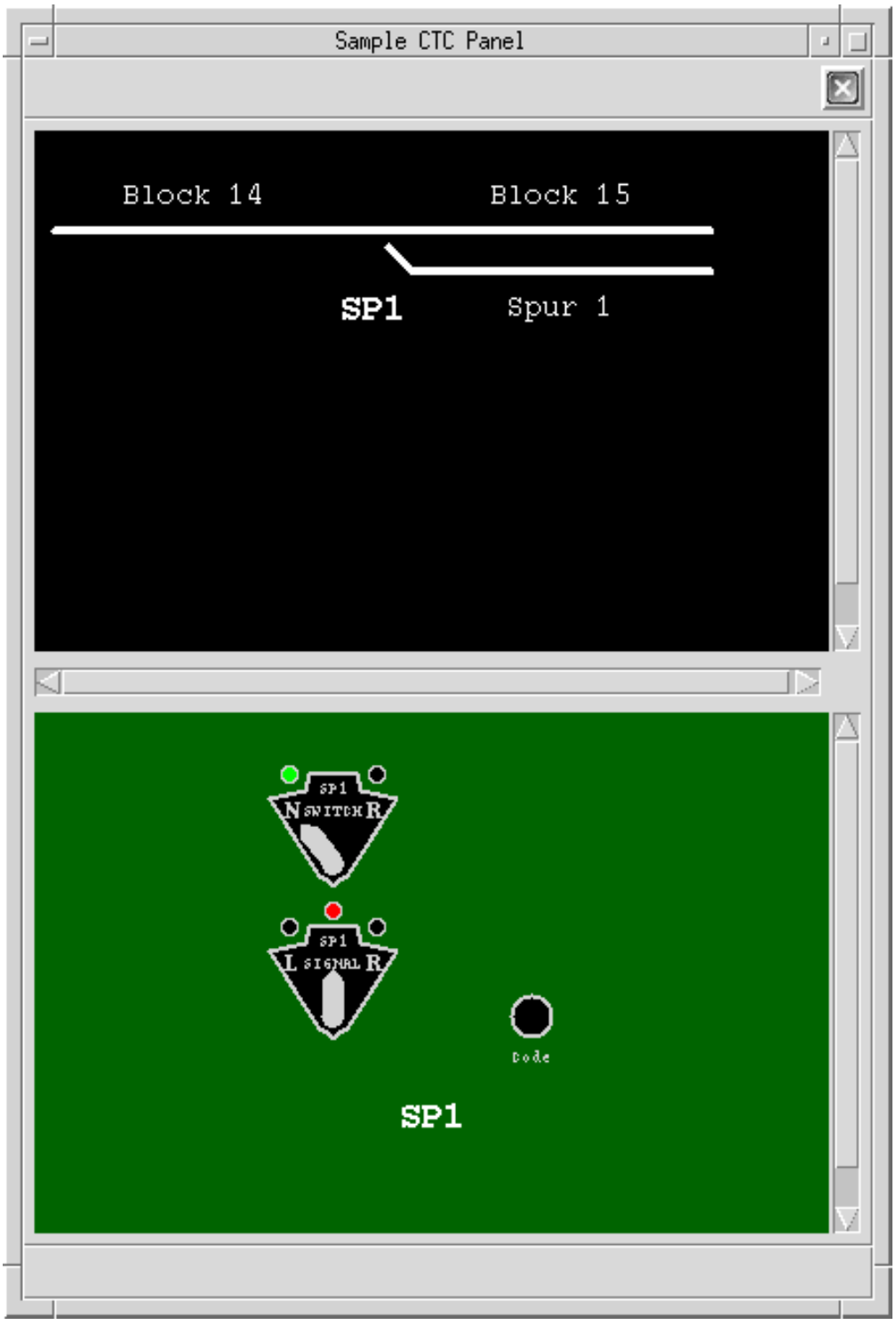

Figure 11.1: Sample CTC Panel Screen Shot

is created at line 61 Lines 67–192 populate the panel with both control panel elements and track schematic elements. The panel is shown in Figure 11.1.

This CTC Panel features a switch (turnout) off the main line to an industrial spur. There is one actual control point, named SP1, and three blocks, two for the mainline blocks and one for the industrial spur. I have put each block into a separate logical control point as a organizational convenience. The control point SP1 contains one piece of trackwork, the switch, and four control panel items, a switch plate, a signal plate, a code button, and a label. The code does not interact with an actual layout. Normally, the −occupiedcommand options of the trackwork elements would define code to read block occupancy detectors and the −statecommand of the switch would define code to read the switch's point position feedback sensors[2]. The code for the switch plate (procedure SetSwitchSP1) would just set the output bits to activate the switch's switch machine. The signal plate code (procedure SetSignalSP1) would also set the output bits to update the aspects of the signals protecting the switch.

[2]Using the CMR/I (Bruce Chubb) Interface, as described in Chapter 5, for example.

Chapter 12

Managing preferences or configuration options

The ReadConfiguration package provides functions for reading, writing, and editing preferences or configuration options. The procedure ReadConfiguration::ReadConfiguration reads in a configuration or preferences file. The ReadConfiguration::WriteConfiguration procedure writes a configuration or preferences file. and the snit macro ReadConfiguration::ConfigurationType creates a snit type to hold a set of configuration options or preferences.

12.1 Configuration or preferences files

Each logical line can contain either an unnamed value (an anonoymous configuration option) or a name value pair. The value can be either a scalar value or a brace enclosed list of alternating keys and values. Tcl bracing, quoting, and comment conventions are used in this file.

Both ReadConfiguration::ReadConfiguration and ReadConfiguration::WriteConfiguration take the same two arguments–a filename and the name of an array variable to receive or provide the preferences.

12.2 A complete Snit type object to hold and manage preferences

Listing 12.1: Creating a configuration object

```
#*
#*

package require ReadConfiguration;#      Load configuration code

namespace eval SampleCode::Configuration {
  # Create the configuration object.
  snit::type Configuration {
    ReadConfiguration::ConfigurationType \
      {{Scratch Folder} scratchfolder directory {~/.scratchfolder} {}} \
      {{Layout Name} {layout name} string {My Layout} {}} \
      {{Layout Scale} {layout scale} enumerated {H0} {Z N H0 0 G}} \
      {{Layout Width} {layout width} integer 96 {10 200}} \
      {{Layout Height} {layout height} integer 48 {10 200}} \
      {{Background Color} backgroundcolor color white {}}
  }
}

proc SampleCode::Configuration::SampleCodeRC {} {
  # Add menu items
  $::SampleCode::Main menu add options command \
        -label "Edit Configuration" \
        -command "::SampleCode::Configuration::Configuration edit"
  $::SampleCode::Main menu add options command \
        -label "Save Configuration" \
        -command "::SampleCode::Configuration::Configuration save"
  $::SampleCode::Main menu add options command \
        -label "Load Configuration" \
        -command "::SampleCode::Configuration::Configuration load"
}

package provide SampleCodeRC 1.0
```

The macro ReadConfiguration::ConfigurationType can be used to create an ensemble command to hold the preferences or configuration options for your program. This macro generates a Snit type that include type methods to load, save, edit, and access preference or configuration options. Typical code to use this macro is shown in Listing 12.1 and the dialog box this macro generates is shown in Figure 12.1.

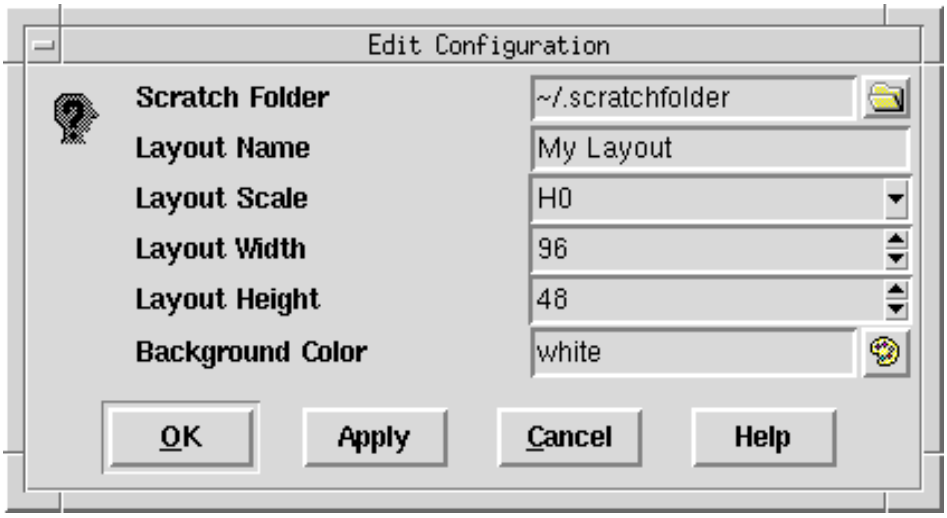

Figure 12.1: Editing a set of preferences

Chapter 13

Using the Graphics Support Code.

Version 2 of the Graphics Support is covered in this chapter. Version 1 is depreciated and is included only for older applications.

The grsupport package defines the constants π and $\frac{\pi}{2}$, a procedure to convert degrees to radians, plus a collection of Snit macros that define common validating methods that are useful when defining Snit types.

Listing 13.1: Graphics Support package examples

```
package require snit;#  Load the snit package
package require Tk;#    Load the Tk package
package require tile;# BWidget Code
package require grsupport 2.0;# Load V2 Graphics Support package.

snit::type Circle {
  # Snit type for drawing circles.

  #*********************
  # Type variables:
  #*********************
  typevariable _MenuId 0;# Ever increasing id for menus

  #*********************
  # Type methods:
  #*********************
  typemethod _GenerateMenuPath {} {
    # Create a unique menu path.
    incr _MenuId
    return "circleMenu$_MenuId"
  }
  typemethod bindtocanvas {canvas sequence {extracode {}}} {
```

```
    # Bind circle creation to a canvas.
    bind $canvas $sequence \
        [mytypemethod _CreateCircle \
                      $canvas %x %y "$extracode"]
  }
  typemethod _CreateCircle {canvas mx my extracode} {
    $type create %%AUTO%% $canvas \
                -x [$canvas canvasx $mx] \
                -y [$canvas canvasy $my] \
                -extracode "$extracode"
  }

  #*********************
  # Methods:
  #*********************
  GRSupport::VerifyDoubleMethod;# Verify double valued options.
  GRSupport::VerifyColorMethod;#  Verify color valued options.
  method _ConfigureXY {option value} {
  # Method to configure X or Y.
  # <in> option The name of the option to configure.
  # <in> value The new value.
  # [index] _ConfigureXY!method

    set oldx $options(-x)
    set oldy $options(-y)
    set options($option) $value
    set dx [expr {$options(-x) - $oldx}]
    set dy [expr {$options(-y) - $oldy}]
    $canvas move $selfns $dx $dy
    set x $options(-x)
    set y $options(-y)
    set size $options(-size)
    set sx [expr {$x + $size}]
    set sy [expr {$y + $size}]
    set centerX [expr {$x + ($size * 0.5)}]
    set centerY [expr {$y + ($size * 0.5)}]
  }
  method _ConfigureSize {option value} {
  # Method to configure size.
  # <in> option The name of the option to configure.
  # <in> value The new value.
  # [index] _ConfigureSize!method

    set deltaSize [expr {double($value) / double($options($option))}]
    set options($option) $value
```

```
      $canvas scale $selfns $centerX $centerY \
                                $deltaSize $deltaSize
70    foreach {x y sx sy} [$canvas coords $selfns] {break}
      set options(-x) $x
      set options(-y) $y
    }
    method _ConfigureFillColor {option value} {
75  # Method to configure a fill color.
    # <in> option The name of the option to configure.
    # <in> value The new value.
    # [index] _ConfigureFillColor!method

80    set options($option) $value
      set  tag $selfns
      catch {$canvas itemconfigure ${tag}$option -fill "$value"}
    }
    method _ConfigureOutlineColor {option value} {
85  # Method to configure an outline color.
    # <in> option The name of the option to configure.
    # <in> value The new value.
    # [index] _ConfigureOutlineColor!method

90    set options($option) $value
      set  tag $selfns
      catch {$canvas itemconfigure ${tag}$option -outline "$value"}
    }
    method Circumfrence {} {
95    # Method to return the circumfrence of the circle.
      # Uses PI from the Graphics Support library.
      return [expr {$options(-size) * $::GRSupport::PI}]
    }
    # Methods bound to events for circles: resizing, moving, and a popup menu
100 # for changing colors and deleting
    method _StartResize {mx my} {
      set rx [expr {[$canvas canvasx $mx] - $centerX}]
      set ry [expr {[$canvas canvasy $my] - $centerY}]
      set Rdxy [expr {sqrt($rx*$rx + $ry*$ry)}]
105   set RorgSize $options(-size)
    }
    method _Resize {mx my} {
      set x [expr {[$canvas canvasx $mx] - $centerX}]
      set y [expr {[$canvas canvasy $my] - $centerY}]
110   set dxy [expr {sqrt($x*$x + $y*$y)}]
      set dsize [expr {($dxy - $Rdxy)*2.0}]
      $self configure -size [expr {$RorgSize + $dsize}]
```

```
    if {[string length "$options(-extracode)"] > 0} {
        uplevel #0 "$options(-extracode)"
    }
  }
  method _StopResize {mx my} {
    catch {unset Rdxy}
    catch {unset RorgSize}
  }
  method _StartMove {mx my} {
    set Mx [expr {[$canvas canvasx $mx] - $options(-x)}]
    set My [expr {[$canvas canvasy $my] - $options(-y)}]
  }
  method _Move {mx my} {
    set x [expr {[$canvas canvasx $mx] - $Mx}]
    set y [expr {[$canvas canvasy $my] - $My}]
    $self configure -x $x -y $y
    if {[string length "$options(-extracode)"] > 0} {
        uplevel #0 "$options(-extracode)"
    }
  }
  method _StopMove {mx my} {
    catch {unset Mx}
    catch {unset My}
  }
  method _PostMenu {X Y} {
    $menupath post $X $Y
  }
  method _setColor {opt menulab} {
    set newcolor [SelectColor $colormenupath -parent $canvas\
                      -color "$options($opt)" -type popup]
    if {[string length "$newcolor"] > 0} {
      $self configure $opt "$newcolor"
      $menupath entryconfigure "$menulab" -foreground "$newcolor"
    }
    $menupath unpost
  }
  method _delete {} {
    $self destroy
  }
  #*************************
  # Variables:
  #*************************
  variable canvas;#     Canvas the circle is drawn on.
  variable sx;#         Right side of circle.
  variable sy;#         Bottom side of circle.
```

```
    variable centerX;#     Center of circle (X).
    variable centerY;#     Center of circle (Y).
    variable menupath;#    Our popup menu
    variable colormenupath;#    Our ColorSelect popup menu
    variable Rdxy;#        Resizing variable
    variable RorgSize;#    Resizing variable
    variable Mx;#          Moving variable
    variable My;#          Moving variable

    #**************************
    # Options:
    #**************************
    # Upper left corner (x,y).
    option -x -default 0 -validatemethod _VerifyDouble \
                          -configuremethod _ConfigureXY
    option -y -default 0 -validatemethod _VerifyDouble \
                         -configuremethod _ConfigureXY
    # Size (diameter) of circle.
    option -size -default 100 -validatemethod _VerifyDouble \
                              -configuremethod _ConfigureSize
    # Color of the circle.
    option -outline -default black -validatemethod _VerifyColor \
                                   -configuremethod _ConfigureOutlineColor
    option -fill -default black -validatemethod _VerifyColor \
                                -configuremethod _ConfigureFillColor
    # Extra code to run after creating, moving, resizing, and deleting circles.
    option -extracode -default {} -readonly yes
    #**************************
    # Constructor: draw the circle.
    #**************************
    constructor {_canvas args} {
      set canvas $_canvas
      set tag $selfns
      catch {$canvas delete $tag}
      $self configurelist $args
      set x $options(-x)
      set y $options(-y)
      set size $options(-size)
      set sx [expr {$x + $size}]
      set sy [expr {$y + $size}]

      set menupath $canvas.[$type _GenerateMenuPath]
      set colormenupath $canvas.colormenu$_MenuId
      menu $menupath -tearoff no
      $menupath add command -label {Fill Color} \
```

```
                                -foreground "$options(-fill)" \
                                -command [mymethod _setColor \
                                                -fill {Fill Color}]
    $menupath add command -label {Outline Color} \
                                -foreground "$options(-outline)" \
                                -command [mymethod _setColor \
                                                -outline {Outline Color}]
    $menupath add command -label {Delete} \
                                -command [mymethod _delete]
    bind $menupath <Escape> "$menupath unpost"

    $canvas create oval $x $y $sx $sy \
                -outline "$options(-outline)" \
                -fill    "$options(-fill)" \
                -width   2 \
                -tag     [list $tag ${tag}-fill \
                                ${tag}-outline]
    $canvas bind $tag <Shift-ButtonPress-2> \
                            [mymethod _StartResize %x %y]
    $canvas bind $tag <Shift-Button2-Motion> \
                            [mymethod _Resize %x %y]
    $canvas bind $tag <Shift-ButtonRelease-2> \
                            [mymethod _StopResize %x %y]
    $canvas bind $tag <ButtonPress-2> \
                            [mymethod _StartMove %x %y]
    $canvas bind $tag <Button2-Motion> \
                            [mymethod _Move %x %y]
    $canvas bind $tag <ButtonRelease-2> \
                            [mymethod _StopMove %x %y]
    $canvas bind $tag <ButtonPress-3> \
                            [mymethod _PostMenu %X %Y]
    if {[string length "$options(-extracode)"] > 0} {
        uplevel #0 "$options(-extracode)"
    }
}
#**************************
# Descructor: remove the circle.
#**************************
destructor {
    catch {$canvas delete $selfns}
    catch {destroy $menupath}
    if {[string length "$options(-extracode)"] > 0} {
        uplevel #0 "$options(-extracode)"
    }
}
```

```
}
250

package provide SampleCodeCircle 1.0
```

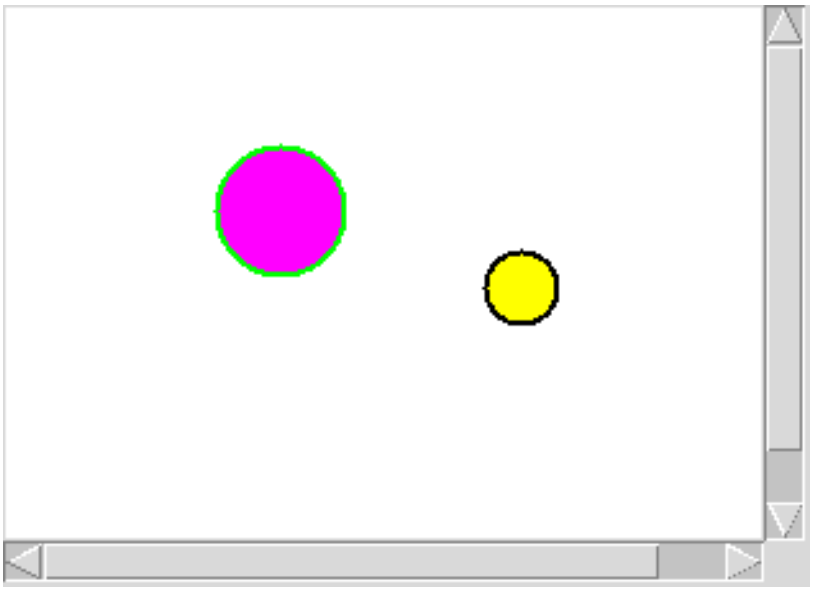

Figure 13.1: Circles drawn with the Circle type.

Listing 13.1 shows some code that uses the Graphics Support package. This code defines a Snit type named Circle, which extends the oval item type for canvas drawing. The type draws an oval, and then binds several events to the oval. Figure 13.1 shows a pair of circles drawn with the code in Listing 13.1.

This code uses several constants and macros defined in the Graphics Support package, including code to verify option values and the constant π.

Chapter 14

Using the Panel Instruments Widgets

Listing 14.1: Using the Panel Instrument based widgets

```
# $Id: SampleCodeInstrumentPanel.tcl 1709 2014-05-21 16:02:04Z heller $

package require Instruments 2.0

namespace eval SampleCode {
}
proc SampleCode::ToggleInstrumentPanelSlide {} {
    variable InstrumentPanelSlideState
    variable Main
    $Main slideout $InstrumentPanelSlideState instrumentPanel
}
proc SampleCode::UpdateRTClock {} {
    variable Clock
    scan [::clock format [::clock scan now] -format %R] \
        {%2d:%2d} hour minute
    $Clock settime $hour  $minute
    after 60000 SampleCode::UpdateRTClock
}
proc SampleCode::UpdateFastClock {} {
    variable fastHours
    variable fastMinutes
    variable realMillisecsPerFastMinute
    variable FastClock
    incr fastMinutes
    if {$fastMinutes > 59} {
```

```
        incr fastHours
        set fastMinutes 0
        if {$fastHours > 12} {
          set fastHours 1
        }
      }
      $FastClock settime $fastHours $fastMinutes
      after $realMillisecsPerFastMinute SampleCode::UpdateFastClock
}

proc SampleCode::SampleCodeInstrumentPanel {} {
  variable Main
  variable InstrumentPanelSlide [$Main slideout add instrumentPanel]
  variable InstrumentPanelSlideState hide
  $Main menu add view checkbutton -label "Instrument Panel" \
        -variable ::SampleCode::InstrumentPanelSlideState \
        -onvalue show -offvalue hide \
        -command ::SampleCode::ToggleInstrumentPanelSlide
  set sw [ScrolledWindow $InstrumentPanelSlide.sw \
                -scrollbar both -auto both]
  pack $sw -expand yes -fill both
  variable InstrumentPanelCanvas [canvas $sw.canvas \
                                        -width 200 \
                                        -height 500 \
                                        -borderwidth 0 \
                                        -relief flat \
                                        -background white]
  $sw setwidget $InstrumentPanelCanvas

  variable Voltmeter [Instruments::DialInstrument \
        create volts \
        $InstrumentPanelCanvas -x 5 -y 5 -size 90 \
        -maxvalue 20 -label "Track Volts" \
        -scaleticksinterval 1]

  variable Ampmeter [Instruments::DialInstrument \
        create amps \
        $InstrumentPanelCanvas -x 105 -y 5 -size 90 \
        -maxvalue 10 -label "Track Amps" \
        -scaleticksinterval 1]

  variable Clock [Instruments::AnalogClock \
        create clock $InstrumentPanelCanvas \
                -x 5 -y 125 -size 90 \
                -label {Real Time}]
```

```
    variable FastClock [Instruments::AnalogClock \
74          create fastclock $InstrumentPanelCanvas \
                -x 105 -y 125 -size 90 \
                -label {Fast Clock}]

    UpdateRTClock
79  variable fastHours 12
    variable fastMinutes 0
    variable FTFactor 8
    variable realMillisecsPerFastMinute [expr {(60*1000)/$FTFactor}]
    UpdateFastClock
84 }

package provide SampleCodeInstrumentPanel 1.0
```

Listing 14.1 contains code to create some of the panel instrument widgets. The window this code helped to create is shown in Figure 14.1. Several typical panel instruments are shown with the code to create them.

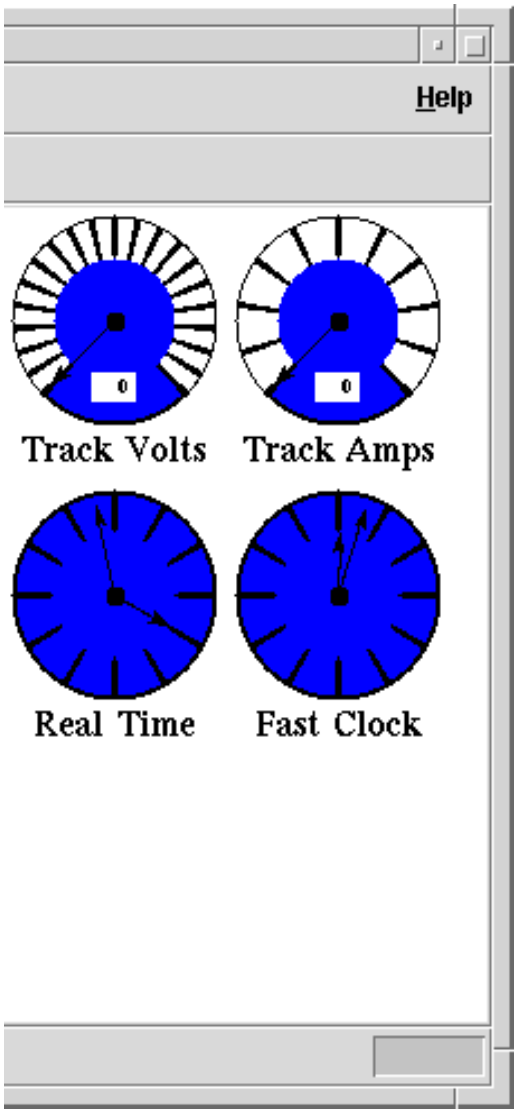

Figure 14.1: Some Panel Instruments

Chapter 15

Using the Oval Widgets

Chapter 16

Various LCARS Widgets

Chapter 17

Creating StarKits and StarPacks

Listing 17.1: Makefile fragment for building a StarPack

```
                SampleCode_Version.h
        -rm -rf SampleCode/
        $(DOXYGEN) Doxyfile.SampleCode
        -rm -f SampleCode/pages.html

EXTRA_DIST += SampleCode.tcl $(SampleCodePackages) SampleCode.h \
                Doxyfile.SampleCode.in

SampleCode${EXEEXT}: SampleCode.tcl $(SampleCodeScripts) $(SampleCodeHelpDep) \
                $(SampleCodePackages)
        -rm -rf SampleCode.kit
        -rm -rf SampleCode.vfs
        $(TCLKIT) $(top_srcdir)/BuildScripts/sdx.kit qwrap $(srcdir)/SampleCode.tcl
        $(TCLKIT) $(top_srcdir)/BuildScripts/sdx.kit unwrap SampleCode.kit
        -rm -rf SampleCode.kit
        $(UNZIP) -qq -d SampleCode.vfs/lib \
                $(top_srcdir)/BuildData/bwidget-1.9.6.zip
        $(TCLKIT) $(top_srcdir)/BuildScripts/AddKitDir.kit SampleCode \
                lib $(SNITLIB)
        $(TCLKIT) $(top_srcdir)/BuildScripts/AddKitFile.kit SampleCode \
                lib/Common $(SampleCodeScripts)
        $(TCLKIT) $(top_srcdir)/BuildScripts/MakePkgIndex.kit SampleCode \
                Common
```

The Makefile fragment shown in Listing 17.1 builds a StarPack. It starts by removing any existing kit or kit directory[1]. Then it uses the SDX qwrap and unwrap

[1] Basicly, cleaning up from a previously failed build.

commands to create a base level kit directory. It then uses helper kits to populate this directory with additional scripts and data files, then finally wraps the resulting directory into a StarPack with the SDX wrap command (the –runtime option makes this a StarPack rather than a StarKit).

17.1 Helper programs included with the Model Railroad System

Three tclkits are included with the Model Railroad System to help you build your own programs using elements of the Model Railroad System Tcl/Tk library of packages:

AddKitDir.kit This kit adds a symbolic link to a directory in a kit directory tree. This is a "lazy" copy of a directory that is usually a directory containing a Tcl/Tk package.

AddKitFile.kit This kit adds one or more files to a directory in a kit directory tree.

MakePkgIndex.kit This kit runs the Tcl command pkg_mkIndex on a directory under the lib directory in a kit directory tree.

These tclkits help with some of the common tasks involved in build StarKits and StarPacks.

17.1.1 AddKitDir.kit – Add a directory to a StarKit or StarPack

This kit does a "lazy" copy[2] of a directory to a StarKit's or StarPack's vfs[3] directory. It takes three parameters: the name of the kit (without the .kit or .vfs extension), the relative path to the parent directory, and the path to the directory to add. A symbolic link, using the tail of the directory path to add, is made in the StarKit's or StarPack's .vfs directory under the relative path specified.

Listing 17.2: Adding a directory to a kit.

```
tclkit AddKitDir.kit MyProgram lib /usr/share/bwidget1.8.0
```

[2]Simply a symbolic link.
[3]Virtual File System.

The example in Listing 17.2 adds a symbolic link named bwidget1.8.0 to /usr/share/bwidget1.8.0 in MyProgram.vfs/lib/. When the kit is wrapped, the sdx program will make a deep copy of this directory into the resultant kit or pack.

17.1.2 AddKitFile.kit – Add files to a StarKit or StarPack

This kit copies one or more files to a directory in a StarKit's or StarPack's vfs directory. It takes three or more parameters: the name of the kit (without the .kit or .vfs extension), the relative path to directory to add files to[4], and one or more files to copy.

Listing 17.3: Adding files to a kit.

```
tclkit AddKitFile.kit MyProgram lib/packages mypackage1.tcl \
                        mypackage2.tcl myextension.so
```

The example in Listing 17.3 adds the files mypackage1.tcl, mypackage2.tcl, and myextension.so to the directory MyProgram.vfs/lib/packages.

17.1.3 MakePkgIndex.kit – Create a pkgIndex.tcl file for a directory in a StarKit or StarPack

This kit runs pkg_mkIndex on a directory under the lib directory in a StarKit's or StarPack's vfs directory. It takes two parameters: the name of the kit (without the .kit or .vfs extension) and the directory under the lib directory to run pkg_mkIndex over.

Listing 17.4: Creating the pkgIndex.tcl for a directory of packages in a kit.

```
tclkit MakePkgIndex.kit MyProgram packages
```

The example in Listing 17.4 runs pkg_mkIndex over the directory MyProgram.vfs/lib/packages.

[4]The directory is created if it does not already exist.

Bibliography

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

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

[3] Bruce Chubb. Ustpqb.txt. On the web at the URL: `http://www.jlcenterprises.net/Files/USTPQB.TXT`., 2004.

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

Index