```
#!/usr/bin/perl
# ============================================================================
# FILE: DnB.pl                                                    10-17-2020
#
# SERVICES: D&B Model Railroad Control Program
#
# DESCRIPTION:
#   This program is used to automate operations on the D&B HO scale model
#   railroad. This Raspberry Pi based program and associated electronics
#   replaces the Parallax Basic Stamp based control system. Refer to the
#   program help and the following for documentation related to this new
#   control system.
#
#   Notebook: D&B Model Railroad Raspberry Pi Control
#   Webpage: http://www.buczynski.com/DnB_rr/DnB_Rpi_Overview.html
#
#   For information on the Basic Stamp version, refer to the following.
#
#   Notebook: D&B Basic Stamp
#   Webpage: http://www.buczynski.com/DnB_rr/DnB_Overview.shtml
#
#   This program is written for perl on Raspberry Pi 3.
#
# PERL VERSION: 5.24.1
#
#      (c) Copyright (c) 2018  Don Buczynski.  All Rights Reserved.
# ============================================================================
use strict;

# ----------------------------------------------------------------------------
# The begin block is used to add the directory holding the DnB perl modules
# to the perl search path. In the process, a couple of global variables are
# defined.

BEGIN {
   use Cwd;
   our $WorkingDir = cwd();
   our ($ExecutableName) = ($0 =~ /([^\/\\]*)$/);
   if (length($ExecutableName) != length($0)) {
      $WorkingDir = substr($0, 0, rindex($0, "/"));
   }
   unshift (@INC, $WorkingDir);
   srand;      # Initialize random number seed.
}

# ----------------------------------------------------------------------------
# External module definitions.
use Getopt::Std;
use Forks::Super;
use Forks::Super::Debug;
use DnB_Mainline;
use DnB_Sensor;
use DnB_Signal;
use DnB_Turnout;
use DnB_GradeCrossing;
use DnB_Yard;
use DnB_Webserver;
use DnB_Message;
use DnB_Simulate;
use POSIX qw(:signal_h :errno_h :sys_wait_h);
```

```perl
use RPi::WiringPi;
use RPi::Const qw(:all);
use Time::HiRes qw(sleep);
use File::Copy;

# -------------------------------------------------------------------------
# Global variables.
our $WorkingDir;                       # CLI working directory.
our $ExecutableName;                   # CLI program name;
our %Opt = ();                         # CLI options storage
our $DebugLevel = -1;                  # Debug level, set by -d option.
our $MainRun = 0;                      # Main program flag.
                                       #    1 = Initialize complete.
                                       #    2 = Simulation active.
                                       #    3 = Main loop active.

our $ChildName = 'Main';               # Name of child process, for ctrl+c.

our $SerialDev = '/dev/serial0';       # Default serial port device
our $SerialBaud = 115200;              # Default baud rate;
our $SerialPort = "";                  # Set if serial port is open.

our $SoundPlayer = "/usr/bin/aplay -q -N -f cd $WorkingDir/wav";
our $AudioVolume = 80;                 # Default audio volume.

our $WebRootDir = '/home/pi/perl/web'; # Webserver document root directory.
our $ListenPort = 8080;                # Webserver port.
our $WebDataDir = '/dev/shm';          # Webserver data exchange directory.

my $TurnoutFile = join("/", $WorkingDir, "TurnoutDataFile.txt");
my $Shutdown = 0;                      # Set to 1 when shutdown button is pressed.

# ==============================================================================
# DnB Program Start/Stop
#
# DnB.pl is a perl program that runs under Raspbian operating system; a Debian
# based Linux specifically developed for the Raspberry Pi. To prevent possible
# corruption of the SD card software, the OS should be properly shutdown prior
# to removing power from the Raspberry Pi. Further, the software is designed to
# start and run "headless"; that it, without interaction with the Linux CLI for
# normal operations. The following describes these processes.
#
# Startup:
#
# The /etc/rc.local file is used to automatically launch DnB.pl once Linux has
# completed boot. (Attempts to use systemd for startup were unsuccessful, the
# program was always killed.) Configure rc.local using the CLI as follows.
#
#    1. sudo nano /etc/rc.local
#    2. Add the following to the file just before the exit 0 line. Change the
#       path to the DnB.pl file if stored in a different place.
#
#       /home/pi/perl/DnB.pl -w -q
#
#    3. Use ^O and ^X editor commands to save and exit.
#
# Note: '>> /dev/shm/DnB.log 2>&1' could be used in place of -q to send the
#       DnB.pl console output to a log file. The log file could then be
#       monitored using 'tail -f /dev/shm/DnB.log' in a seperate command window.
#       Use a path in /home/pi if the log needs to be retained when the RPi is
```

```
#       powered down.
#
# During startup, if the shutdown button is held down, the DnB.pl program will
# acknowledge the button hold and exit startup. The RPi will then be usable
# for normal Raspbian CLI/GUI interaction using monitor, mouse, and keyboard.
# Use the CLI/GUI from this point to shutdown Raspbian.
#
# Shutdown:
#
# A momentary contact button is connected across GPIO21 and ground. GPIO21 is
# configured as an input with pullup enabled. This circuit is monitored by
# DnB.pl. Detection of a button press initiates a 10 second delay during which
# five tones will be sounded. During the delay period, shutdown can be aborted
# by another press of the shutdown button. At the end of the delay, the main
# program performs an orderly shutdown of the software processes and places the
# hardware interfaces into a 'safe condition'. The Raspbian OS will then be
# shutdown.
#
# Safe condition serves to help protect the servos, sound modules, and signal
# lamps should layout power remain on for an extended period. The following
# shutdown steps are performed.
#
#    1. Stop all child processes.
#    2. Raise crossing gates and semaphore flag board.
#    3. Wait for in-progress turnout moves to complete.
#    4. Turn off all servo channels.
#    5. Turn off all signal lamps.
#    6. Turn off all GPIO driven relays and indicator lamps.
#    7. Turn off holdover indicator lamps.
#    8. Save the current servo positions to TurnoutDataFile.txt.
#    9. Shutdown Raspbian OS using:  sudo shutdown -h now
#
# Once the Raspberry Pi green activity LED is no longer flashing, about 10-15
# seconds, it is safe to power off the layout electronics.

# ==============================================================================
# RPi Sound Player
#
# All sound wave files are output, using the $SoundPlayer variable definition,
# by the PlaySound subroutine located in DnB_Yard. The PCM playback volume is
# set to default -1800 (max = 400, min = -10000) during startup. This value
# can be changed using the -v command line option.

# ==============================================================================
# Turnout Related Data
#
# The ServoBoardAddress hash holds the I2C address of the servo driver boards.
# It is used to populate the 'Addr' entries in the %TurnoutData hash.

our %ServoBoardAddress = ('1' => 0x41, '2' => 0x42);

# The TurnoutData hash stores the information used to position the turnouts on
# the layout. The storage structure is known as a 'hash-of-hashes'. This type
# of data structure simplifys access by the code. Only a pointer to the hash
# is needed when communicating the dataset to code blocks.
#
#  %TurnoutData (
#     Turnout1 => {                                   Value
#        Pid => <pid of forked MoveTurnout process>     0
#        Addr => <driver_board_I2C_address>,            -
```

```
#        Port => <driver_board_servo_port>,             -
#        Pos => <last_servo_pwm_position>,             600
#        Rate => <servo_move_rate_pwm_per_sec>,        450
#        Open => <turnout_open_pwm_value>,             350
#        Middle => <turnout_middle_pwm_value>,         600
#        Close => <turnout_close_pwm_value>,           850
#        MinPos => <minimum_servo_pwm_position>,       300
#        MaxPos => <maximum_servo_pwm_position>,       900
#        Id => <Identification string>                  -
#     },
#     Turnout2 => {
#        ...
#     }
#  );
#
# The following initializes the hash with default data. Default data is used
# to write the initial TurnoutDataFile contents. Thereafter, these values are
# overwritten during program startup by TurnoutDataFile file load. This allows
# the user to change the operating values for Rate, Open, Close, and Min/Max
# for layout needs. Note, the name keys are case sensitive. A 'Rate' value of
# 450 moves the turnout servo from Open (350) to Close (850) in 1.1 seconds.
#
# Once the servo mechanical adjustments and operational servo positions are
# determined using the TurnoutDataFile file, those values should be entered
# into the %TurnoutData hash below. This ensures that if the TurnoutDataFile
# file is regenerated using the -f option, the operational position values
# will be preserved.
#
# Important: The ~100 hz PCA9685 refresh rate calculation in I2C_InitServoDriver
#            results in MinPos:300 and MaxPos:900 for the SG90 servo. When
#            adjusting turnout point positions, do not exceed these limits.
#            The values shown above will result in full servo motion and
#            rotational rate.
#
# %TurnoutData{'00'} is used for temperature related processing. The ambient
# room temperature, in degrees C, is read from a DS18B20 temperature sensor.
# The Timeout variable is used by the main loop to periodically update the
# temperature value; every 5 minutes. The temperature value is used in the
# MoveTurnout code to apply a position adjustment to the semaphore and gate
# servos. This helps to counteract for thermal expansion/contraction of the
# layout benchwork. The mechanical signal devices are sensitive to this effect.

my %TurnoutData = (
   '00' => {'Temperature' => 0, 'Timeout' => 0},
   '01' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 0,
            'Pos' => 540, 'Rate' => 200, 'Open' => 640, 'Middle' => 590,
            'Close' => 540, 'MinPos' => 535, 'MaxPos' => 645,
            'Id' => 'Mainline turnout T01'},
   '02' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 1,
            'Pos' => 545, 'Rate' => 200, 'Open' => 643, 'Middle' => 600,
            'Close' => 545, 'MinPos' => 540, 'MaxPos' => 648,
            'Id' => 'Mainline turnout T02'},
   '03' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 2,
            'Pos' => 620, 'Rate' => 200, 'Open' => 510, 'Middle' => 570,
            'Close' => 620, 'MinPos' => 505, 'MaxPos' => 625,
            'Id' => 'Mainline turnout T03'},
   '04' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 3,
            'Pos' => 600, 'Rate' => 450, 'Open' => 350, 'Middle' => 600,
            'Close' => 850, 'MinPos' => 300, 'MaxPos' => 900,
            'Id' => 'spare'},
```

```
    '05' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 4,
             'Pos' => 555, 'Rate' => 200, 'Open' => 555, 'Middle' => 610,
             'Close' => 660, 'MinPos' => 550, 'MaxPos' => 665,
             'Id' => 'Mainline turnout T05'},
    '06' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 5,
             'Pos' => 550, 'Rate' => 200, 'Open' => 650, 'Middle' => 600,
             'Close' => 550, 'MinPos' => 545, 'MaxPos' => 655,
             'Id' => 'Mainline turnout T06'},
    '07' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 6,
             'Pos' => 495, 'Rate' => 200, 'Open' => 615, 'Middle' => 560,
             'Close' => 462, 'MinPos' => 457, 'MaxPos' => 620,
             'Id' => 'Mainline turnout T07'},
    '08' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 7,
             'Pos' => 520, 'Rate' => 200, 'Open' => 670, 'Middle' => 600,
             'Close' => 520, 'MinPos' => 515, 'MaxPos' => 675,
             'Id' => 'Yard turnout T08'},
    '09' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 8,
             'Pos' => 625, 'Rate' => 200, 'Open' => 495, 'Middle' => 570,
             'Close' => 625, 'MinPos' => 490, 'MaxPos' => 630,
             'Id' => 'Yard turnout T09'},
    '10' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 9,
             'Pos' => 545, 'Rate' => 200, 'Open' => 675, 'Middle' => 615,
             'Close' => 545, 'MinPos' => 540, 'MaxPos' => 680,
             'Id' => 'Yard turnout T10'},
    '11' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 10,
             'Pos' => 550, 'Rate' => 200, 'Open' => 650, 'Middle' => 600,
             'Close' => 550, 'MinPos' => 545, 'MaxPos' => 655,
             'Id' => 'Yard turnout T11'},
    '12' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 11,
             'Pos' => 705, 'Rate' => 200, 'Open' => 570, 'Middle' => 620,
             'Close' => 705, 'MinPos' => 565, 'MaxPos' => 710,
             'Id' => 'Yard turnout T12'},
    '13' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 12,
             'Pos' => 655, 'Rate' => 200, 'Open' => 500, 'Middle' => 580,
             'Close' => 655, 'MinPos' => 495, 'MaxPos' => 660,
             'Id' => 'Yard turnout T13'},
    '14' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 13,
             'Pos' => 650, 'Rate' => 200, 'Open' => 480, 'Middle' => 560,
             'Close' => 650, 'MinPos' => 475, 'MaxPos' => 655,
             'Id' => 'Yard turnout T14'},
    '15' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 14,
             'Pos' => 630, 'Rate' => 200, 'Open' => 480, 'Middle' => 550,
             'Close' => 630, 'MinPos' => 475, 'MaxPos' => 635,
             'Id' => 'Yard turnout T15'},
    '16' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'1'}, 'Port' => 15,
             'Pos' => 705, 'Rate' => 200, 'Open' => 555, 'Middle' => 620,
             'Close' => 705, 'MinPos' => 550, 'MaxPos' => 710,
             'Id' => 'Yard turnout T16'},
    '17' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 0,
             'Pos' => 680, 'Rate' => 200, 'Open' => 530, 'Middle' => 610,
             'Close' => 680, 'MinPos' => 525, 'MaxPos' => 685,
             'Id' => 'Yard turnout T17'},
    '18' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 1,
             'Pos' => 695, 'Rate' => 200, 'Open' => 550, 'Middle' => 620,
             'Close' => 695, 'MinPos' => 545, 'MaxPos' => 700,
             'Id' => 'Yard turnout T18'},
    '19' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 2,
             'Pos' => 715, 'Rate' => 200, 'Open' => 540, 'Middle' => 620,
             'Close' => 715, 'MinPos' => 535, 'MaxPos' => 720,
             'Id' => 'Yard turnout T19'},
```

```
   '20' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 3,
            'Pos' => 620, 'Rate' => 200, 'Open' => 495, 'Middle' => 550,
            'Close' => 620, 'MinPos' => 490, 'MaxPos' => 625,
            'Id' => 'Yard turnout T20'},
   '21' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 4,
            'Pos' => 520, 'Rate' => 200, 'Open' => 670, 'Middle' => 600,
            'Close' => 520, 'MinPos' => 515, 'MaxPos' => 675,
            'Id' => 'Yard turnout T21'},
   '22' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 5,
            'Pos' => 600, 'Rate' => 200, 'Open' => 440, 'Middle' => 520,
            'Close' => 595, 'MinPos' => 435, 'MaxPos' => 600,
            'Id' => 'Yard turnout T22'},
   '23' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 6,
            'Pos' => 525, 'Rate' => 200, 'Open' => 675, 'Middle' => 600,
            'Close' => 525, 'MinPos' => 520, 'MaxPos' => 680,
            'Id' => 'Yard turnout T23'},
   '24' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 7,
            'Pos' => 520, 'Rate' => 200, 'Open' => 670, 'Middle' => 600,
            'Close' => 520, 'MinPos' => 515, 'MaxPos' => 675,
            'Id' => 'Yard turnout T24'},
   '25' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 8,
            'Pos' => 490, 'Rate' => 200, 'Open' => 630, 'Middle' => 560,
            'Close' => 490, 'MinPos' => 485, 'MaxPos' => 635,
            'Id' => 'Yard turnout T25'},
   '26' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 9,
            'Pos' => 480, 'Rate' => 200, 'Open' => 645, 'Middle' => 560,
            'Close' => 480, 'MinPos' => 475, 'MaxPos' => 650,
            'Id' => 'TT turnout T26'},
   '27' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 10,
            'Pos' => 670, 'Rate' => 200, 'Open' => 670, 'Middle' => 590,
            'Close' => 515, 'MinPos' => 510, 'MaxPos' => 675,
            'Id' => 'TT turnout T27'},
   '28' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 11,
            'Pos' => 600, 'Rate' => 450, 'Open' => 350, 'Middle' => 600,
            'Close' => 850, 'MinPos' => 300, 'MaxPos' => 900,
            'Id' => 'spare'},
   '29' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 12,
            'Pos' => 600, 'Rate' => 450, 'Open' => 350, 'Middle' => 600,
            'Close' => 850, 'MinPos' => 300, 'MaxPos' => 900,
            'Id' => 'spare'},
   '30' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 13,
            'Pos' => 525, 'Rate' => 75, 'Open' => 520, 'Middle' => 600,
            'Close' => 675, 'MinPos' => 515, 'MaxPos' => 690,
            'Id' => 'Semaphore'},
   '31' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 14,
            'Pos' => 765, 'Rate' => 65, 'Open' => 765, 'Middle' => 705,
            'Close' => 635, 'MinPos' => 625, 'MaxPos' => 775,
            'Id' => 'GC02 Gate 1 (near)'},
   '32' => {'Pid' => 0, 'Addr' => $ServoBoardAddress{'2'}, 'Port' => 15,
            'Pos' => 490, 'Rate' => 65, 'Open' => 490, 'Middle' => 555,
            'Close' => 620, 'MinPos' => 480, 'MaxPos' => 630,
            'Id' => 'GC02 Gate 2 (far)'});

# Since MoveTurnout is a slow process, each turnout position change is forked
# to prevent blocking the main program. A simple fork does not support passing
# of child data back to the parent. Since the final turnout position is needed
# from the child, a 'piped fork' is used. At fork activation, the child process
# pipes STDOUT and STDERR are mapped to the TurnoutData hash, 'Pos' and 'Pid'
# respectively, for the turnout being moved.
#
```

```
# Following fork activation, the child process pid is stored in the TurnoutData
# hash for the turnout. The turnout move is 'inprogress' until this pid value is
# again zero. The child process prints the final turnout position to STDOUT and
# zero to STDERR. These values are written directly to the %TurnoutData hash due
# to the pipe configurations set at activation.

# ==============================================================================
# Signal Related Data
#
# - Track Plan -
#
# When reduced to simplest form, the DnB trackplan consists of the following
# electrical blocks (Bxx) and searchlight signals (Lxx). The character < or >
# shows the train direction controlled (or lamp reflectors if you want to think
# of it that way).

#                         L03>     <L04             L09>     <L10
#   /==B01==\        <L02 /====B04====\        <L08 /====B07====\====\
#            =====B03=====             =====B06=====            B09  B10
#   \==B02==/ L01>        \====B05====/ L07>        \====B08====/====/
#                         L05>     <L06             L11>     <L12

# The following rules are used to illuminate the signals.
#
#   Signal        Condition
#   ------        ---------
#    Off          Unoccupied block not being approached
#   Green         Approaching unoccupied block
#    Red          Approaching occupied block
#   Yellow        Approaching unoccupied block; subsequent block occupied

# - Signal Control -
#
# The GpioData hash holds the Raspberry Pi GPIO pin data that is used to access
# the driver hardware controlling the layout signals and power polarity relays.
# The pins are manipulated by RPi::WiringPi to communicate with the 74HC595 shift
# register which in turn drives the signal LEDs. The power polarity relays are
# driven directly by the GPIO pins. The Init_SignalDriver code creates the
# necessary pin objects and stores the object pointer in this hash.

# GPIO set to hardware PWM mode.

my %GpioData = (
   'GPIO17_XLAT' => {'Desc' => '74HC595 Data Latch', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO23_OUTE' => {'Desc' => '74HC595 Output Enable', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO27_SCLK' => {'Desc' => '74HC595 Serial Clock', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO22_DATA' => {'Desc' => '74HC595 Data', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO5_PR01'  => {'Desc' => 'Power Polarity relay 01', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO6_PR02'  => {'Desc' => 'Power Polarity relay 02', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO13_PR03' => {'Desc' => 'Power Polarity relay 03', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO19_FE01' => {'Desc' => 'Keypad 01 first entry LED', 'Mode' => 1,
                     'Obj' => 0},
   'GPIO26_HLCK' => {'Desc' => 'Holdover route lock LED', 'Mode' => 1,
```

```
                       'Obj' => 0},
   'GPIO20_TEST' => {'Desc' => 'Timing Test signal', 'Mode' => 1,
                       'Obj' => 0},
   'GPIO21_SHDN' => {'Desc' => 'Shutdown button', 'Mode' => 0,
                       'Obj' => 0});

# Note: GPIO4 is reserved for use by the DS18B20 temperature sensor. It is
#       accessed/controlled using Raspbian modprobe 1-wire protocol. Refer
#       to Turnout.pm GetTemperature for configuration details.

# RPi::Pin needs numeric value inputs. Defenitions are as follows.
#    'Mode': 0=Input, 1=Output, 2=PWM_OUT, 3=GPIO_CLOCK

# The SignalData hash stores information about the signals. Each entry uses a
# consecutive pair of bits in the shift register; bits 0 and 1 for signal 1,
# bits 2 and 3 for signal 2, etc. A bicolor LED is wired across the bit pair
# and illuminates red for one voltage polarity (e.g. bit 0 high, bit 1 low) and
# green for the opposite polarity (bit 0 low, bit 1 high). If both bits are the
# same state, high or low, the LED is off. This provides the needed signal
# states; off, red, and green. The specific state for each signal is determined
# by the code using the block detector inputs.

# The color yellow is achieved by rapidly switching a signal between red
# and green. The human eye perceives this action as the color yellow. The
# SignalChildProcess performs this by using two internal shift register
# buffers. Yellow signals are red in one and green in the other. The buffers
# are alternately sent to the shift register.

# The bits associated with SignalData 13 and 14 are used for the grade crossing
# signals. See the 'Grade Crossing Data' section below for information as to
# how these bits are utilized.

my %SignalData = (
   '01' => {'Bits' => '0,1',   'Current' => 'Off', 'Desc' => 'Track B3 control'},
   '02' => {'Bits' => '2,3',   'Current' => 'Off', 'Desc' => 'Track B3 control'},
   '03' => {'Bits' => '4,5',   'Current' => 'Off', 'Desc' => 'Track B4 control'},
   '04' => {'Bits' => '6,7',   'Current' => 'Off', 'Desc' => 'Track B4 control'},
   '05' => {'Bits' => '8,9',   'Current' => 'Off', 'Desc' => 'Track B5 control'},
   '06' => {'Bits' => '10,11', 'Current' => 'Off', 'Desc' => 'Track B5 control'},
   '07' => {'Bits' => '12,13', 'Current' => 'Off', 'Desc' => 'Track B6 control'},
   '08' => {'Bits' => '14,15', 'Current' => 'Off', 'Desc' => 'Track B6 control'},
   '09' => {'Bits' => '16,17', 'Current' => 'Off', 'Desc' => 'Track B7 control'},
   '10' => {'Bits' => '18,19', 'Current' => 'Off', 'Desc' => 'Track B7 control'},
   '11' => {'Bits' => '20,11', 'Current' => 'Off', 'Desc' => 'Track B8 control'},
   '12' => {'Bits' => '22,23', 'Current' => 'Off', 'Desc' => 'Track B8 control'},
   '13' => {'Bits' => '24,25', 'Current' => 'Off', 'Desc' => 'GC 1 LEDs'},
   '14' => {'Bits' => '26,27', 'Current' => 'Off', 'Desc' => 'GC 2 LEDs'},
   '15' => {'Bits' => '28,29', 'Current' => 'Off', 'Desc' => 'Unused'},
   '16' => {'Bits' => '30,31', 'Current' => 'Off', 'Desc' => 'Unused'});

# The algorithm used for setting a signal's color is based upon the track plan
# and signalling rules. Each track block, when occupied by a train, results in
# a set of signal indications as described in the %SignalColor hash. The color
# values are derrived by assuming a single occupied track block.

#                                    Signal
# ActiveBlock   S01  S02  S03  S04  S05  S06  S07  S08  S09  S10  S11  S12
# -----------   ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---
#    B01        GRN  YEL
#    B02        GRN  YEL
```

```
#    B03        RED  RED  GRN  YEL  GRN  YEL
#    B04        YEL  GRN  RED  RED            GRN  YEL
#    B05        YEL  GRN            RED  RED  GRN  YEL
#    B06                  YEL  GRN  YEL  GRN  RED  RED  GRN  YEL  GRN  YEL
#    B07                                      YEL  GRN  RED  RED
#    B08                                      YEL  GRN            RED  RED
#    B09                                                YEL  GRN  YEL  GRN
#    B10                                                YEL  GRN  YEL  GRN

# When multiple track blocks are occupied, color priority is applied since a
# signal might have more than one color indication. For example, if only B03
# is occupied, the color for S03 is green. If both B03 and B04 are occupied,
# S03 could be either green or red. The correct color to display is red. When
# a signal would display more than one color, the following color priority is
# used: Red = highest, Yellow = medium, Green = lowest.

# To accomplish this prioritization, the block sensor inputs are processed three
# times, 1st for green indications, 2nd for yellow indications, and lastly for
# red indications. In this way, red overwrites green or yellow, and yellow
# overwrites green.

# Primary key maps to the %SensorBit hash. The secondary key maps to the
# %SignalData hash.

my %SignalColor = (
   '00' => {'Grn' => '01',          'Yel' => '02'},
   '01' => {'Grn' => '01',          'Yel' => '02'},
   '02' => {'Grn' => '03,05',       'Yel' => '04,06',       'Red' => '01,02'},
   '03' => {'Grn' => '02,07',       'Yel' => '01,08',       'Red' => '03,04'},
   '04' => {'Grn' => '02,07',       'Yel' => '01,08',       'Red' => '05,06'},
   '05' => {'Grn' => '04,06,09,11', 'Yel' => '03,05,10,12', 'Red' => '07,08'},
   '06' => {'Grn' => '08',          'Yel' => '07',          'Red' => '09,10'},
   '07' => {'Grn' => '08',          'Yel' => '07',          'Red' => '11,12'},
   '08' => {'Grn' => '10,12',       'Yel' => '09,11'},
   '09' => {'Grn' => '10,12',       'Yel' => '09,11'});

# - Semaphore Signal -
#
# The SemaphoreData hash holds information related to the Semaphore signal. This
# signal is modeled as the old style moveable flag board semaphore. The lamp in
# this signal is a low voltage incandescent bulb. The lamp is driven by a bit of
# the associated signal bit pair defined in the SignalData hash. The SignalData
# primary key (signal number) is the primary key used in the SemaphoreData hash.

my %SemaphoreData = (
   '08' => {'Servo' => '30', 'InMotion' => 0, 'Lamp' => 'Off'});

# The SemaphoreData hash identifies the TurnoutData servo used with the signal.
# The 'Open' (green), 'Middle' (yellow) and 'Closed' (red) positions of the flag
# board can be adjusted like the turnouts by modifying the TurnoutDataFile file.

# The SemaphoreData hash is also used to persist control data. Due to signal flag
# board motion, multiple calls to the SetSemaphoreSignal code will occur before a
# previously requested color position is completed.

# When setting signal colors, the main code checks the SemaphoreData hash for the
# signal being processed. If present, the SetSemaphoreSignal routine us used for
# setting its color. See the description of this code for further information.

# ==============================================================================
```

```
# Grade Crossing Data
#
# There are two grade crossings on the DnB model railroad, each with flashing
# signals and one with crossing gates. Across-the-track infrared sensors are
# used to detect train presence. These sensors are mapped to bit positions in
# the %SensorBit hash. At program startup, a dedicated child process is started
# for each grade crossing. The child process is used to handle the signal lamp
# flashing. Gate positioning, and logic to send the 'start' and 'stop' commands
# to the signal child process, is handled by the ProcessGradeCrossing code.

my %GradeCrossingData = (
   '00' => {'WebUpdate' => 0},
   '01' => {'Pid' => 0,                  # Pid of child process.
            'SigPid' => 0,               # Pid of SignalChildProcess.
            'AprEast' => '10',           # %SensorBit east approach sensor bit.
            'Road' => '11',              # %SensorBit road sensor bit.
            'AprWest' => '12',           # %SensorBit west approach sensor bit.
            'Signal' => '13',            # %SignalData lamp bits.
            'Gate' => '',                # %TurnoutData gate servo(s).
            'State' => 'idle',           # Current grade crossing state.
            'AprTimer' => 0,             # Approach activity timer.
            'RoadTimer' => 0,            # Road activity timer.
            'DepTimer' => 0,             # Departure activity timer.
            'SigRun' => 'off',           # Signal lamps active.
            'GateDelay' => 0,            # Not used, no gates for this signal.
            'GateServo' => 0,            # Not used, no gates for this signal.
            'SoundApr' => '4,GPIOB4',    # Approach sound GPIO control bit.
            'SoundRoad' => '4,GPIOB5'    # Road sound GPIO control bit.
           },
   '02' => {'Pid' => 0,                  # Pid of child process.
            'SigPid' => 0,               # Pid of SignalChildProcess.
            'AprEast' => '13',           # %SensorBit east approach sensor bit.
            'Road' => '14',              # %SensorBit road sensor bit.
            'AprWest' => '15',           # %SensorBit west approach sensor bit.
            'Signal' => '14',            # %SignalData lamp bits.
            'Gate' => '31,32',           # %TurnoutData gate servo(s).
            'State' => 'idle',           # Current grade crossing state.
            'AprTimer' => 0,             # Approach activity timer.
            'RoadTimer' => 0,            # Road activity timer.
            'DepTimer' => 0,             # Departure activity timer.
            'SigRun' => 'off',           # Signal lamps active.
            'GateDelay' => 0,            # Working delay for 'gateLower' state.
            'GateServo' => 0,            # Working servo for 'gateRaise' state.
            'SoundApr' => '4,GPIOB6',    # Approach sound GPIO control bit.
            'SoundRoad' => '4,GPIOB7'    # Road sound GPIO control bit.
           });

# The Signal number maps to the SignalData hash. The grade crossing signals are
# wired to red only Leds, one to each bit of the signal position. When the signal
# is set to 'Red', one lamp will illuminate. When set to 'Grn', the other signal
# Led illuminates. When set to 'Off' both Leds are off. This methodology saves
# 74HC595 shift register bits and facilitates the use of common signal color code.

# The grade crossing with gates maps to the %TurnoutData hash for controlling the
# associated servos. A lowered gate is set by using the 'Close' parameter value
# in the %TurnoutData hash. A raised gate uses the 'Open' parameter value. Adjust
# these values as needed to achieve the desired motion, rate, and end positions.

# Both signals have an associated sound module which produces grade crossing bell
# sound effects. The sound effects are switched on/off by output GPIO bits on a
```

```
# sensor board as identified by the 'Sound' parameter in the GradeCrossingData
# hash identifies. The first GPIO activates the 'bell only' sound and the second
# GPIO activates the 'bell + train noise' sound.
#
# Note that the second sound is not used due to sound1/sound2 switching issues
# related to these old sound modules.

# =============================================================================
# Sensor Related Data
#
# The SensorChip hash holds the I2C addresses of the I/O PI Plus boards. The
# mapped address is applied to the sensors referenced in the %SensorBit hash
# below. Each I/O Pi Plus board has twp MCP23017 chips. Each chip has two con-
# figurable 8 bit ports. The sensor initialization code establishes an object
# reference for each chip and stores it in this hash for later use to read
# sensor input. Hash entries DirA (direction PortA), DirB (direction PortB),
# PolA (bit polarity PortA), PolB (bit polarity PortB), PupA (pullup enable
# PortA), and PupB (pullup enable PortB) are used only for chip initialization.
# See MCP23017 data sheet for details.

my %SensorChip = (
   '1' => {'Addr' => 0x20, 'Obj' => 0, 'DirA' => 0xFF, 'DirB' => 0xFF,
           'PolA' => 0x00, 'PolB' => 0xFC, 'PupA' => 0x00, 'PupB' => 0x00},
   '2' => {'Addr' => 0x21, 'Obj' => 0, 'DirA' => 0xFF, 'DirB' => 0xFF,
           'PolA' => 0xFF, 'PolB' => 0xFF, 'PupA' => 0x00, 'PupB' => 0x00},
   '3' => {'Addr' => 0x22, 'Obj' => 0, 'DirA' => 0xC3, 'DirB' => 0xC3,
           'PolA' => 0x00, 'PolB' => 0x00, 'PupA' => 0xC3, 'PupB' => 0xC3},
   '4' => {'Addr' => 0x23, 'Obj' => 0, 'DirA' => 0xFF, 'DirB' => 0x00,
           'PolA' => 0xFF, 'PolB' => 0x00, 'PupA' => 0xFF, 'PupB' => 0x00});

# The MCP23017 has a number of internal registers that are used to read the
# sensor inputs. These registers are defined as follows. See the MCP23017
# data sheet for usage information. These register addresses are dependent on
# IOCON.BANK being set to 0. (Established by I2C_InitSensorDriver).

my %MCP23017 = (
   'IODIRA' => 0x00, 'IODIRB' => 0x01, 'IOPOLA' => 0x02, 'IOPOLB' => 0x03,
   'IOCON'  => 0x0A, 'GPPUA' => 0x0C,  'GPPUB' => 0x0D,  'GPIOA'  => 0x12,
   'GPIOB' =>  0x13, 'OLATA' => 0x14,  'OLATB' => 0x15);

# The SensorState hash stores the active track sensor information associated
# with chips 1 and 2. The program periodically reads each sensor chip and
# sets this hash accordingly.

my %SensorState = ('1' => 0, '2' => 0);

# There are 16 bits per MCP23017 chip. They are defined in the SensorBit hash.

my %SensorBit = (
   '00' => {'Chip' => '1', 'Bit' => 'GPIOA0', 'Desc' => 'Block detector B01'},
   '01' => {'Chip' => '1', 'Bit' => 'GPIOA1', 'Desc' => 'Block detector B02'},
   '02' => {'Chip' => '1', 'Bit' => 'GPIOA2', 'Desc' => 'Block detector B03'},
   '03' => {'Chip' => '1', 'Bit' => 'GPIOA3', 'Desc' => 'Block detector B04'},
   '04' => {'Chip' => '1', 'Bit' => 'GPIOA4', 'Desc' => 'Block detector B05'},
   '05' => {'Chip' => '1', 'Bit' => 'GPIOA5', 'Desc' => 'Block detector B06'},
   '06' => {'Chip' => '1', 'Bit' => 'GPIOA6', 'Desc' => 'Block detector B07'},
   '07' => {'Chip' => '1', 'Bit' => 'GPIOA7', 'Desc' => 'Block detector B08'},
   '08' => {'Chip' => '1', 'Bit' => 'GPIOB0', 'Desc' => 'Block detector B09'},
   '09' => {'Chip' => '1', 'Bit' => 'GPIOB1', 'Desc' => 'Block detector B10'},
   '10' => {'Chip' => '1', 'Bit' => 'GPIOB2', 'Desc' => 'GC1 AprEast'},
```

```
   '11' => {'Chip' => '1', 'Bit' => 'GPIOB3', 'Desc' => 'GC1 Road'},
   '12' => {'Chip' => '1', 'Bit' => 'GPIOB4', 'Desc' => 'GC1 AprWest'},
   '13' => {'Chip' => '1', 'Bit' => 'GPIOB5', 'Desc' => 'GC2 AprEast'},
   '14' => {'Chip' => '1', 'Bit' => 'GPIOB6', 'Desc' => 'GC2 Road'},
   '15' => {'Chip' => '1', 'Bit' => 'GPIOB7', 'Desc' => 'GC2 AprWest'},
   '16' => {'Chip' => '2', 'Bit' => 'GPIOA0', 'Desc' => 'Sensor S01 (B3 T01)'},
   '17' => {'Chip' => '2', 'Bit' => 'GPIOA1', 'Desc' => 'Sensor S02 (B2 exit)'},
   '18' => {'Chip' => '2', 'Bit' => 'GPIOA2', 'Desc' => 'Sensor S03 (B1 exit)'},
   '19' => {'Chip' => '2', 'Bit' => 'GPIOA3', 'Desc' => 'Sensor S04 (spare)'},
   '20' => {'Chip' => '2', 'Bit' => 'GPIOA4', 'Desc' => 'Sensor S05 (B4 T05)'},
   '21' => {'Chip' => '2', 'Bit' => 'GPIOA5', 'Desc' => 'Sensor S06 (B5 T06)'},
   '22' => {'Chip' => '2', 'Bit' => 'GPIOA6', 'Desc' => 'Sensor S07 (B6 T07)'},
   '23' => {'Chip' => '2', 'Bit' => 'GPIOA7', 'Desc' => 'Sensor S08 (B7 T07)'},
   '24' => {'Chip' => '2', 'Bit' => 'GPIOB0', 'Desc' => 'Sensor S09 (B8 T07)'},
   '25' => {'Chip' => '2', 'Bit' => 'GPIOB1', 'Desc' => 'Sensor S10 (B1 Yel)'},
   '26' => {'Chip' => '2', 'Bit' => 'GPIOB2', 'Desc' => 'Sensor S11 (B1 Red)'},
   '27' => {'Chip' => '2', 'Bit' => 'GPIOB3', 'Desc' => 'Sensor S12 (B2 Yel)'},
   '28' => {'Chip' => '2', 'Bit' => 'GPIOB4', 'Desc' => 'Sensor S13 (B2 Red)'},
   '29' => {'Chip' => '2', 'Bit' => 'GPIOB5', 'Desc' => 'Unused'},
   '30' => {'Chip' => '2', 'Bit' => 'GPIOB6', 'Desc' => 'Unused'},
   '31' => {'Chip' => '2', 'Bit' => 'GPIOB7', 'Desc' => 'Unused'});

# The hidden holdover tracks employ sensors which are used to indicate train
# position in the B01 and B02 blocks. These sensors are located close to the
# exit end of these blocks. The sensors drive yellow and red panel LEDs. As
# a train approaches the S2 and S3 sensors, first the yellow and then the red
# LED will begin to flash. In this way, the engineer can stop the train prior
# to activating the S2/S3 sensor; which causes the holdover turnouts to be
# set for holdover departure. The %PositionLed hash holds the LED information
# that is used by the PositionChildProcess code. The primary index of this
# hash maps to the primary index in the %SensorBit hash.

my %PositionLed = (
   '25' => {'Chip' => '4', 'Bit' => 'GPIOB0', 'Olat' => 'OLATB',
            'Desc' => 'B01 yellow LED'},
   '26' => {'Chip' => '4', 'Bit' => 'GPIOB1', 'Olat' => 'OLATB',
            'Desc' => 'B01 red LED'},
   '27' => {'Chip' => '4', 'Bit' => 'GPIOB2', 'Olat' => 'OLATB',
            'Desc' => 'B02 yellow LED'},
   '28' => {'Chip' => '4', 'Bit' => 'GPIOB3', 'Olat' => 'OLATB',
            'Desc' => 'B02 red LED'});

# ==============================================================================
# Track Plan: Reverse Loop and Hold-over Tracks

# The trackage involved with this section is hidden and used for train trip
# hold-over and return. Two sidings are available each with a train presence
# block detector (Bx), track power polarity reverse relay (Px), and optical
# sensors (Sx) to detect train movement. Three turnouts (Tx) are used to move
# trains in and out of this section.

#            ----------------- B1/P1 -----------------
#           /                                          \
#          /  ---------------- B2/P2 ----------------   \
#     r1 /   / r2                                r3 \   \ r4
#        | |                                         | |
#        \ | S2                                      | / S3
#         \|                                         |/
#       T2 \                                         / T3
#            -------------------    -----------------
```

```
#                              \  /
#                            T1 |/
#                               | S1
#                               |
#                               B3
#                               |
#                               ~

# Reverse loop operation requires that for an inbound or outbound operation,
# with respect to a siding, the rail polarity must match mainline rail polarity.
# This rail polarity match is required only while power drawing portions of a
# train are in transit across the siding rail gaps.

# In operation, a train on the mainline approaches the reverse loop. It is
# detected by sensor S1. If block detector B1 is inactive, T1, T2, and P1 are
# set to direct the train to siding B1. If block detector B1 is active, T1, T3,
# and P2 are set to direct the train to siding B2. If B2 is also active, the
# train wreck warning is sounded. Turnouts are used this way to take advantage
# of the 'straight' side of hidden turnouts T2 and T3 to minimize derailments.
# Trains always move clockwise through siding B1 and counter-clockwise through
# siding B2.

# A train leaving B1 or B2 will be detected by S3 or S2 respectively. T1/T3/P1
# or T1/T2/P2 are set to direct the train back onto the mainline.

# For an inbound or outbound operation, it is necessary to disable acting on S1
# active indications following the initial one. For the inbound direction, this
# prevents turnouts from changing as the block detector B1/B2 begins reporting
# the presence of a train. In the outbound direction, it prevents assumption of
# an inbound train and T1 operation.

# Stopping or backing a inbound or outbound train will have no effect on these
# operations unless the outbound sensor S2 or S3 has been reached. If so, the
# turnouts and block power polarity will be set for an outbound condition and
# incorrectly set for a backup operation. A train should not be backed up once
# it is more than half way into a siding.

# An operational deficiency was noted in this track section. Train movement
# through these turnouts following correction of derailments was troublesome
# due to the automatic turnout positioning. With the RPi design, a four button
# keypad is added to permit route selection. The buttons correspond to the
# routes (r1-r4) leading from the B3 mainline to each end of the B1 and B2
# sidings.

# Following a holdover button press, the three turnouts will be positioned for
# the specified route. A tone will be sounded and an active indicator on the
# keypad will be illuminated. The route will remain active until:
#
#   1. One of the four buttons is pressed.
#   2. No S1, S2, or S3 sensor activity is detected for 30 seconds.

# Track Plan: Midway Sidings

#                                              S5   T5
#          ------------------ B4 ----------------------- B3 -- ~
#         /                                        /
#        /   ---------------- B5 ------------------
#       /   /
#       \  | S6
#        \ |
```

```
#         \| T6
#          |
#          B6
#          |
#          ~

# The track involved with this section provides a place for mainline trains to
# pass each other. The associated turnouts simulate proto typical turnouts that
# are "spring loaded" to a specific position. When entering, the train is always
# directed to a specific track. When exiting, the turnout points are positioned
# to permit train passage. Once the last car of the train passes through the
# turnout, its points are set back to the "normal" position.

# Normal position routes a train approching T5 from B3 to siding B5. A train
# approaching T6 from B6 is routed to siding B4. A train leaving B4 or B5 will
# be detected by sensors S5 or S6 respectively. The points of T5 or T6 will be
# set to direct the train back onto the mainline. A retriggerable timeout is
# used to debounce the S5 and S6 sensor inputs. Three seconds after the last car
# transits the sensor, the turnout is repositioned to "normal".

# Track Plan: Yard Approach Wye

#            --------- B10 -------
#           /                    \
#          /--------- B9 ---------\
#         |                        |
#         B7                       B8
#          \                       /
#           \         P3          /
#            \                   /
#         S8  \                 /  S9
#              \-----   ------/
#                T7  \ /
#                     |
#                     B6
#                     |  S7
#                     ~

# The track involved with this section provides a "wye" turnout; the legs of
# which are approach tracks leading to opposite ends of the yard. This forms a
# reverse loop that includes B7 through B10 and all of the yard tracks. The
# blocks are individual only for the purpose of signaling. Tracks leading to
# and including the yard tracks from T7 are wired to polarity control relay P3.

# Turnout T7 is only partially controlled. The last set route will be used for
# trains in B6 approaching T7 unless manually changed by the train engineer. The
# T7 turnout points will be set automatically for B7 or B8 trains approaching T7
# when detected by S8 or S9. The power polarity relay P3 will be set based on the
# position of T7 to yard track power polarity matches B6 track power.

# In all cases, it is not necessary to "ignore" sensor inputs in either direction
# of travel. Detections by S8 or S9 following S7 will not change T7 or P3 from
# their current states. The same is true for S7 detections following S8 or S9.

# The TrackData hash, primary key sensor number, stores information that is used
# to set turnouts and track power polarity based on train movement that activates
# sensor (Sx) and block (Bx) input.

my %TrackData = (
   '01' => {'Timeout' => 0, 'Last' => 'B2', 'Direction' => 'In',
```

```perl
                'WaitB3Inact' => 0, 'RouteLocked' => 0, 'RouteTime' => 0,
                'Sensor' => 16},
   '02' => {'Timeout' => 0, 'Sensor' => 17},
   '03' => {'Timeout' => 0, 'Sensor' => 18},
   '04' => {0},
   '05' => {'Timeout' => 0, 'Inactive' => 'Open', 'Active' => 'Close',
            'ManualSet' => 0, 'Locked' => 0, 'Sensor' => 20},
   '06' => {'Timeout' => 0, 'Inactive' => 'Close', 'Active' => 'Open',
            'ManualSet' => 0, 'Locked' => 0, 'Sensor' => 21},
   '07' => {'Timeout' => 0, 'Polarity' => 0, 'Sensor' => 22},
   '08' => {'Timeout' => 0},
   '09' => {'Timeout' => 0});

# ==============================================================================
# Keypad User Input
#
# The KeypadData hash holds information related to push button keypad input.
# A 'Storm K Range' 4x4 button keypad matrix is connected to a MCP23017 port.
# Within the keypad, normally open push buttons are connected to the inter-
# section of each row and column. Pressing a button will cause the associated
# row and column to be electrically connected. By driving the columns and
# scanning the rows, the pressed button can be determined.

#     row/col   1   2   3   4
#               |   |   |   |
#       A ------0---1---2---3--
#               |   |   |   |
#       B ------4---5---6---7--
#               |   |   |   |
#       C ------8---9---A---B--
#               |   |   |   |
#       D ------C---D---E---F--
#               |   |   |   |

# See DnB_Sensor::ReadKeypad subroutine for keypad to MCP23017 pin mapping.

my %KeypadData = (
   '01' => {'Chip' => '3', 'Row' => 'GPIOA', 'Col' => 'OLATA', 'Last' => -1,
            'PressTime' => 0, 'Entry1' => -1, 'Gpio' => 'GPIO19_FE01'},
   '02' => {'Chip' => '3', 'Row' => 'GPIOB', 'Col' => 'OLATB', 'Last' => -1,
            'PressTime' => 0, 'Entry1' => -1, 'Gpio' => 'tbd'});

   # Note: MCP23017 chips are initialized by DnB_Sensor::I2C_InitSensorDriver
   #       using the values specified in the %SensorChip hash.

# The first pressed button number will be stored in 'Entry1'. Two button presses
# are needed to set a yard route. 'Gpio' identifies the GPIO used for the keypad
# first entry indicator. If a second button is not entered within 2 seconds, the
# first key press is discarded.

# Non-matrix buttons are identified in the %ButtonData hash. These are single
# bit sized values corresponding to a button press. A 'Storm K Range' 1x4 button
# keypad is connected to a MCP23017 port.

#     button    D   C   B   A
#               |   |   |   |
#               o---o---o---o-- common

# See DnB_Sensor::GetButton subroutine for keypad to MCP23017 pin mapping.
```

```
my %ButtonData = (
   '00' => {'Chip' => '4', 'Bit' => 'GPIOA3', 'Last' => 0,
            'Desc' => 'Turnout T5 toggle', 'PressTime' => 0, 'Turnout1' => '05',
            'Turnout2' => '06'},
   '01' => {'Chip' => '4', 'Bit' => 'GPIOA2', 'Last' => 0,
            'Desc' => 'Turnout T6 toggle', 'PressTime' => 0, 'Turnout1' => '06',
            'Turnout2' => '05'},
   '02' => {'Chip' => '4', 'Bit' => 'GPIOA1', 'Last' => 0,
            'Desc' => 'Turnout T7 open', 'Turnout' => '07'},
   '03' => {'Chip' => '4', 'Bit' => 'GPIOA0', 'Last' => 0,
            'Desc' => 'Turnout T7 close', 'Turnout' => '07'},
   '04' => {'Chip' => '4', 'Bit' => 'GPIOA4', 'Last' => 0,
            'Desc' => 'Request holdover route 1', 'PressTime' => 0},
   '05' => {'Chip' => '4', 'Bit' => 'GPIOA5', 'Last' => 0,
            'Desc' => 'Request holdover route 2', 'PressTime' => 0},
   '06' => {'Chip' => '4', 'Bit' => 'GPIOA6', 'Last' => 0,
            'Desc' => 'Request holdover route 3', 'PressTime' => 0},
   '07' => {'Chip' => '4', 'Bit' => 'GPIOA7', 'Last' => 0,
            'Desc' => 'Request holdover route 4', 'PressTime' => 0},
   'FF' => {'Gpio' => 'GPIO21_SHDN', 'Wait' => 0, 'Shutdown' => 0, 'Step' => 0,
            'Time' => 0, 'Tones' => 'G,F,E,D,C,C_'});

# The T5 and T6 toggle buttons provide for manually toggling the position of
# the respective turnout. This functionality is used for special train operations
# involving this section of track. Button input is ignored if the respective
# turnout is performing an inprogress timing operation. After manually toggling
# T5 or T6 to the "non-normal" position, these turnouts will automatically reset
# to their normal position once the train completes its transit of the turnout.

# Turnouts T5 or T6 can be "locked" into the non-normal position by pressing the
# appropriate turnout toggle button a second time within .5 second of the first
# depression. The turnout will remain in the non-normal position until manually
# set to the normal position using the respective toggle button.
#
# Both T5 and T6 cannot be locked at the same time; a derailment would occur.
# Locking either T5 or T6 permits a train to be stopped on one of the sidings
# for an extended period of time and not interfere with mainline traffic
# movements using the other track. To unlock a turnout, double press the turnout
# toggle button.

# Buttons are provided for manually toggling the position of turnout T7. This
# functionality is used for selecting the desired approach track to the yard.
# Button input is ignored if the Wye retriggerable timeout counter is non-zero
# indicating that a train is transitting the turnout. Manual change will be
# ignored until one second after the last active detection by S7, S8, or S9.

# ================================================================================
# Yard Route Data
#
# The YardRouteData hash holds information used to set the turnouts (Tx) of the
# yard and approach tracks. The following diagrams illustrates the track and
# turnouts involved.
#                             ~
#                             | T7
#                             |\
#    -------------------------- -------------------------------------
#   /                                                                \
#  1              \                   /    /    /   /   /   /     2
# /                \                 /    /    /   /   /   /       \
# \                 \               /    /    /   /   /   /        /
```

```
#  \                 14            15   6    7   8   9   10  11       /
#   \                 \           /    /    /   /   /   /   /        /
#    \   ------- 13 ---\T25   T22/-T23/    /   /   /   /   /        /
#     \                 \       /    /T19-/   /   /   /   /        /
#      \    ---- 12 --T24\-----/ T18/--------/   /T21/   /        /
#       \                          /         T20/--------   T10  /
#     T8 \--------T12\---------T16/--- 5 ---T17/----/T15------/-/ T11
#         \   T9      16                           16        / /
#          \-\---------\T13----------- 4 ------T14/---------/ /
#             \                                              /
#              \---------------------- 3 ----\--------------/ T27
#                                         T26 \            /
#                                              \--- 16 ---/
#
# Yard and approach tracks are assigned a number; 1 through 16. The track
# number corresponds to the numbered keypad buttons. A route is specified
# by keying in two track numbers. The first number entered is the "from"
# track. It is the track currently occupied by the train. The second number
# entered is the "to" track. It is the desired destination track for the
# train. Once both numbers are input, the turnouts for the specified route
# will be set appropriately. Key combinations that do not correspond to a
# valid route will be ignored and an error tone will sound.
#
# Note: The keypad returns 0-F and these numbers are also used in the
# %YardRouteData index keys. These hexadecimal numbers correspond to tracks
# 1-16.
#
# Keying in the same number for the "from" and "to" tracks will set the
# turnouts to route just the specified track. This is useful for the
# following operations.
#
# Track 3-5: Will set all turnouts on these tracks to their normal
#            (straight) position.
# Track 16:  Will open the four turnouts T12 through T15 for a "run
#            around" operation. Consecutive track 16 entry will close
#            all four turnouts.
#
# There are some special cases that must be handled. These involve from/to
# tracks that are dependent on direction. Since direction is not known, the
# code initially sets turnouts for a left to right movement relative to the
# above diagram. If the same from/to command is consecutively entered, the
# right to left movement is set.
#
#    Track 3 to 16:
#       Initial     - T26
#       Consecutive - T27
#    Track 5 to 4:
#       Initial     - T12 and T13
#       Consecutive - T15 and T14
#    Track 4 to 5:
#       Initial     - T14 and T15
#       Consecutive - T13 and T12
#
# Only the turnouts for the selected route will be affected, all other
# turnouts retain their current position. Turnout positions are stored as
# they are set during operations. This information is referenced during
# subsequent operations to skip the setting of turnouts already in the
# proper position.
#
# To facilitate keypad entry, an indicator is positioned on the keypad.
```

```
# This indicator will be illuminate when the first track number is entered.
# It will extinguished when the second track number is entered.
#
# The %YardRouteData primary index is made up of a 'R' and two hexadecimal
# characters. The first character is the "from" track number. The second
# character is the "to" track number. The value for each index is a comma
# separated list of turnout numbers and their required position.

my %YardRouteData = (
   'Control' => {'Inprogress' => 0, 'Route' => "", 'Step' => 0,
                 'RouteTime' => 0},
   'R02' => 'T08:Close,T09:Open',
   'R03' => 'T08:Close,T09:Close,T13:Close,T12:Close',
   'R04' => 'T08:Open',
   'R05' => 'T08:Open,T12:Close,T13:Close,T16:Open,T18:Open,T19:Open,T23:Close',
   'R06' => 'T08:Open,T12:Close,T13:Close,T16:Open,T18:Open,T19:Close',
   'R07' => 'T08:Open,T12:Close,T13:Close,T16:Open,T18:Close',
   'R08' => 'T08:Open,T12:Close,T13:Close,T16:Close,T17:Open,T20:Open,T21:Open',
   'R09' => 'T08:Open,T12:Close,T13:Close,T16:Close,T17:Open,T20:Open,T21:Close',
   'R0A' => 'T08:Open,T12:Close,T13:Close,T16:Close,T17:Open,T20:Close',
   'R0F' => 'T08:Close,T09:Open,T26:Open',
   'R12' => 'T11:Close,T27:Open',
   'R13' => 'T11:Open,T10:Close',
   'R14' => 'T11:Open,T10:Open',
   'R1F' => 'T11:Close,T27:Close',
   'R20' => 'T26:Close,T09:Open,T08:Close',
   'R21' => 'T11:Close,T27:Open,T26:Close',
   'R22' => 'T26:Close,T27:Open',
   'R2F' => 'T26:Open,T27:Open',
   'r2F' => 'T27:Close,T26:Close',
   'R30' => 'T13:Close,T12:Close,T09:Close,T08:Close',
   'R31' => 'T14:Close,T15:Close,T10:Close,T11:Open',
   'R33' => 'T13:Close,T12:Close,T14:Close,T15:Close',
   'R34' => 'T13:Close,T12:Close,T14:Open,T15:Open',
   'r34' => 'T13:Open,T12:Open,T14:Close,T15:Close',
   'R40' => 'T12:Close,T13:Close,T08:Open',
   'R41' => 'T15:Close,T14:Close,T10:Open,T11:Open',
   'R43' => 'T12:Open,T13:Open,T15:Close,T14:Close',
   'r43' => 'T12:Close,T13:Close,T15:Open,T14:Open',
   'R44' => 'T12:Close,T13:Close,T16:Close,T17:Close,T15:Close,T14:Close',
   'R45' => 'T12:Close,T13:Close,T16:Open,T18:Open,T19:Open,T23:Close',
   'R46' => 'T12:Close,T13:Close,T16:Open,T18:Open,T19:Close',
   'R47' => 'T12:Close,T13:Close,T16:Open,T18:Close',
   'R48' => 'T12:Close,T13:Close,T16:Close,T17:Open,T20:Open,T21:Open',
   'R49' => 'T12:Close,T13:Close,T16:Close,T17:Open,T20:Open,T21:Close',
   'R4A' => 'T12:Close,T13:Close,T16:Close,T17:Open,T20:Close',
   'R50' => 'T23:Close,T19:Open,T18:Open,T16:Open,T12:Close,T13:Close,T08:Open',
   'R54' => 'T23:Close,T19:Open,T18:Open,T16:Open,T12:Close,T13:Close',
   'R55' => 'T23:Close,T19:Open,T18:Open',
   'R5B' => 'T23:Open,T22:Close,T24:Close',
   'R5C' => 'T23:Open,T22:Close,T24:Open,T25:Close',
   'R5D' => 'T23:Open,T22:Close,T24:Open,T25:Open',
   'R60' => 'T19:Close,T18:Open,T16:Open,T12:Close,T13:Close,T08:Open',
   'R64' => 'T19:Close,T18:Open,T16:Open,T12:Close,T13:Close',
   'R66' => 'T19:Close,T18:Open',
   'R70' => 'T18:Close,T16:Open,T12:Close,T13:Close,T08:Open',
   'R74' => 'T18:Close,T16:Open,T12:Close,T13:Close',
   'R77' => 'T18:Close',
   'R80' => 'T21:Open,T20:Open,T17:Open,T16:Close,T12:Close,T13:Close,T08:Open',
   'R84' => 'T21:Open,T20:Open,T17:Open,T16:Close,T12:Close,T13:Close',
```

```
    'R88' => 'T21:Open,T20:Open',
    'R90' => 'T21:Close,T20:Open,T17:Open,T16:Close,T12:Close,T13:Close,T08:Open',
    'R94' => 'T21:Close,T20:Open,T17:Open,T16:Close,T12:Close,T13:Close',
    'R99' => 'T21:Close,T20:Open',
    'RA0' => 'T20:Close,T17:Open,T16:Close,T12:Close,T13:Close,T08:Open',
    'RA4' => 'T20:Close,T17:Open,T16:Close,T12:Close,T13:Close',
    'RAA' => 'T20:Close',
    'RB5' => 'T24:Close,T22:Close,T23:Open',
    'RBB' => 'T24:Close',
    'RBE' => 'T24:Close,T22:Open',
    'RC5' => 'T25:Close,T24:Open,T22:Close,T23:Open',
    'RCC' => 'T25:Close',
    'RCE' => 'T25:Close,T24:Open,T22:Open',
    'RD5' => 'T25:Open,T24:Open,T22:Close,T23:Open',
    'RDD' => 'T25:Open',
    'RDE' => 'T25:Open,T24:Open,T22:Open',
    'REB' => 'T22:Open,T24:Close',
    'REC' => 'T22:Open,T24:Open,T25:Close',
    'RED' => 'T22:Open,T24:Open,T25:Open',
    'REE' => 'T22:Open',
    'RF0' => 'T26:Open,T09:Open,T08:Close',
    'RF1' => 'T27:Close,T11:Close',
    'RF2' => 'T26:Open,T27:Open',
    'rF2' => 'T27:Close,T26:Close',
    'RFF' => 'T12:Open,T13:Open,T14:Open,T15:Open',
    'rFF' => 'T12:Close,T13:Close,T14:Close,T15:Close',
    'X02' => 'T08:Close,T09:Open,T26:Close,T27:Open',
    'X12' => 'T11:Close,T27:Open,T26:Close',
    'X03' => 'T08:Close,T09:Close,T13:Close,T12:Close,T14:Close,T15:Close',
    'X13' => 'T11:Open,T10:Close,T14:Close,T15:Close,T13:Close,T12:Close',
    'X04' => 'T08:Open,T12:Close,T13:Close,T16:Close,T17:Close,T15:Close,' .
             'T14:Close',
    'X14' => 'T11:Open,T10:Open,T12:Close,T13:Close,T16:Close,T17:Close,' .
             'T15:Close,T14:Close');

# ==============================================================================
# Simulation Data
#
# The SimulationData hash holds information that is used to simulate the movement
# of a train over the layout when the -a option is specified on the DnB.pl CLI.
# Each hash entry is a step of that movement and consists of sensor values and a
# time period. This hash is populated and used by code in DnB_Simulate.pm.
#
my %SimulationData = ();

# ==============================================================================
# Webserver
#
# A webserver interface is enabled by specifying the -w option. An external web
# browser can then be used to view various layout operational data. The browser
# connection point (IP:Port) is displayed on the console output. The IP value
# is the Rpi hostname or corresponding numeric (xxx.xxx.xxx.xxx). Port value is
# defined by the $ListenPort variable.
#
# The webserver root directory is defined by $WebRootDir; /home/pi/perl/web.
# Static files, e.g. .gif image or .css files, are stored in this directory.
# Dynamically created content is stored and served from $WebDataDir; normally
# defined as /dev/shm (ramdisk).
#
# Operational data is stored in the $WebDataDir directory about once a second.
```

```
# This data is read and used to build the web pages that are displayed in the
# user's browser. This results in minimal overhead to the main loop code. The
# following data files are used.
#
# sensor.dat       (generated by main loop)
#    Sensor: 32 sensor bits as a numeric value.
#       bit position: 1 = active, 0 = idle.
#    Signal: L01=x,L02=x, ... L12=x
#       x = 'Off', 'Grn', 'Yel', or 'Red'.
#    T01=<value1>:<value2>: ... <value8>
#    T02=<value1>:<value2>: ... <value8>
#    ...
#       value order = Pos, Rate, Open, Middle, Close, MinPos, MaxPos, Id
#
# grade.dat        (generated by ProcessGradeCrossing)
#    GC01: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
#    GC02: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
#       <state> = 'idle', 'gateLower', 'approach', 'road', 'gateRaise' or 'depart'
#       <lamps> = 'on' or 'off'.
#       <gates> = 'Open', 'Closed', or '- none -'
#       <sensor> = 1 (active>) or 0 (idle).
#
# The 'Live' web page displays a graphical representation of the layout track
# blocks and signals. Based on sensor input, the main loop stores the names of
# image files to be displayed in the $WebDataDir directory. The track plan is
# divided into three sections to minimize the number of image files that are
# needed to cover all active block combinations.
#
# Active blocks:
#    y-overlay.dat (yard)      blocks B06 - B10.
#    m-overlay.dat (midway)    blocks B03 - B06.
#    h-overlay.dat (holdover)  blocks B01 - B03.
#
# When a request for a *-overlay.dat file is received by the webserver code,
# the requested file is read for the file name to be served. The named image, a
# transparent .png file with appropriate track blocks colored red, is located in
# the $WebRootDir directory. This image file is then sent to the browser where
# it overlays the background image. Browser java-script is used to auto-refresh
# the overlay images every few seconds while the 'Main Live' page is displayed.
#
# The semaphore signals show a colored indication in a similar manner. The Main
# Live page requests a DnB-Lxx-overlay.dat file for each semaphore. Webserver
# code returns the proper color file which overlays the signal head. Overlay
# positioning is accomplished by the CSS rules specified to the browser. These
# overlay objects are included in the java-script auto-refresh cycle.
#
# Two grade crossing signals show a flashing rXr symbol on the Main Live page
# when the grade crossing is not in the idle state.
#
# The Yard Live page works in a similar manner to display the turnout lined yard
# tracks. Six yard track sections and corresponding Yard-Sx-overlay.dat files
# are used.

# ==============================================================================
# Child Processes
#
# A number of the processing functions are performed as child processes to the
# main code. Child process priority (fork os_priority) is used to balance overall
# program flow. For example,
#
```

```
#   * SignalChildProcess is timing sensitive due to the toggling of red/green
#     to produce a yellow signal indication.
#   * Turnout open/close operations would cause main code blocking until the
#     stepping of an inprogress operation completes.
#
# Normal linux priority for a program is 0. os_priority above normal is set with
# a positive value; below normal is set with a negative value.
#
# The ChildProcess hash functions as a dispatch table and is used to launch each
# child process and store its process ID. 'Code' defines the subroutine code to
# be run and 'Opts' defines the associated arguments. 'Opts' is essentially a
# hash that facilitates the use of an alternate form of the Super::Forks call.
#
# During D&B operation, the PIDs are periodically checked. If found inactive, the
# child process is restarted.
#
our %ChildProcess = (

   # Must be started first. SignalChildProcess code is in DnB_Signal.pm.
   '01' => { 'Name' => 'SignalChild', 'Pid' => 0, 'Code' => \&SignalChildProcess,
             'Opt' => { os_priority => 6, child_fh => 'in socket',
                        args => [ \%GpioData ]
                      }
           },

   # 4x4 keypad child process. The stderr handle is used to send key press data
   # from child to parent. The parent must periodically read the key data using:
   # $key = Forks::Super::read_stderr($ChildProcess{'02'}{'Pid'}); The
   # KeypadChildProcess code is in DnB_Sensor.pm.
   '02' => { 'Name' => 'KeypadChild', 'Pid' => 0, 'Code' => \&KeypadChildProcess,
             'Opt' => { os_priority => 4, child_fh => 'err socket',
                        args => [ '01', \%KeypadData, \%MCP23017, \%SensorChip ]
                      }
           },

   # 1x4 button child process. The stderr handle is used to send button press
   # data, defined in %ButtonData, from child to parent. The parent periodically
   # reads the user button input using: $button = Forks::Super::read_stderr(
   # $ChildProcess{'03'}{'Pid'}); ButtonChildProcess code in DnB_Sensor.pm.
   '03' => { 'Name' => 'ButtonChild', 'Pid' => 0, 'Code' => \&ButtonChildProcess,
             'Opt' => { os_priority => 4, child_fh => 'err socket',
                        args => [ \%ButtonData, \%MCP23017, \%SensorChip ]
                      }
           },

   # Holdover position child process. No data is passed between the parent and
   # child. This process reads the holdover position sensors and illuminates the
   # corresponding panel LED when set. The PositionChildProcess code is in
   # DnB_Sensor.pm.
   '04' => { 'Name' => 'PositionChild', 'Pid' => 0, 'Code' => \&PositionChildProcess,
             'Opt' => { os_priority => 0,
                        args => [ \%SensorBit, \%PositionLed, \%SensorChip,
                                  \%MCP23017 ]
                      }
           },

   # Grade crossing child process for each grade crossing. The SignalChild Pid is
   # required and must be already running. The pid is stored in %GradeCrossingData.
   # The GcChildProcess code is in DnB_GradeCrossing.pm.
   '05' => { 'Name' => 'GcChild 01', 'Pid' => 0, 'Code' => \&GcChildProcess,
```

```
                'Opt' => { os_priority => 2, child_fh => 'in socket',
                           args => [ '01', \%SignalData, \%GradeCrossingData,
                                     \%SensorChip, \%MCP23017 ]
                         }
           },
    '06' => { 'Name' => 'GcChild 02', 'Pid' => 0, 'Code' => \&GcChildProcess,
                'Opt' => { os_priority => 2, child_fh => 'in socket',
                           args => [ '02', \%SignalData, \%GradeCrossingData,
                                     \%SensorChip, \%MCP23017 ]
                         }
           },

    # Webserver process if -w enabled. Webserver code is in DnB_Webserver.pm.
    '07' => { 'Name' => 'WebserverChild', 'Pid' => 0, 'Code' => \&Webserver,
                'Opt' => { os_priority => -1,
                           args => [ $WebRootDir, $ListenPort, $WebDataDir ]
                         }
           }
);

# ===============================================================================

my $UsageText = (qq(
===== Help for $ExecutableName ================================================
This program is used to automate operations on the D&B HO scale model railroad.
This Raspberry Pi based program and associated electronics replaces the Parallax
Basic Stamp based control system. Refer to the following for details.

Notebook: D&B Model Railroad, Raspberry Pi Control
Webpage: http://www.buczynski.com/DnB_rr/DnB_Rpi_Overview.html

For information on the Basic Stamp version, refer to the following.

Notebook: D&B Basic Stamp
Webpage: http://www.buczynski.com/DnB_rr/DnB_Overview.shtml

This program is coded in perl and runs under the Raspbian OS. The RPI::WiringPi
perl module, written by Steve Bertrand, interfaces the various Raspberry Pi
hardware functions, e.g. serial communication and GPIO, with perl.

The shutdown button must be used to properly shutdown the Raspbian OS prior to
removing power from the layout electronics. This is important to prevent possible
corruption of the SD card software. It is safe to power off the electronics once
the green activity LED on the end of the lower board, the Raspberry Pi, does not
flash for about 5 seconds. The DnB program can be safely terminated using ctrl+c
when manually started from the command line.

The DnB.pl program is configured to start automatically as part of Raspbian OS
boot. Hold down the shutdown button prior to, and during power-on to cause the
DnB program to terminate without OS shutdown.

The Raspberry Pi serial port can be used to communicate messages to a monitor
terminal. A USB->COM device such as the Adafruit P954 cable, which also performs
level shifting, is used to connect the Pi to an external computer running a
terminal emulator program. e.g. PuTTy or terraterm. GPIO pin connections on the
Pi end are: 6 (Gnd, blk), 8 (Txd, wht), 10 (Rxd, grn). Set terminal emulator to
115200,8,N,1 for the COM port being used on the USB end.

This control system uses SG90 hobby servos to better model proto-typical turnout
movement. Two Adafruit I2C 16-Channel servo boards are used. The individual servo
```

```
positions are controlled by the pulse width values set in these driver boards by
the DnB program. Last position information for each turnout is saved as part of
normal shutdown. It is used for servo positioning on the subsequent power up or
program restart. The crossing gate and semaphore servos are also controlled
through by these driver boards.

The file holding the servo position information, TurnoutDataFile.txt, can be user
modified using a text editor. Typically, the 'Open', 'Close', and 'Rate' values
are adjusted for the desired turnout operation. The changed values will be used
on the subsequent program start. Should the file become hopelessly corrupt, it
can be restored to defaults using the -f option. A backup of the existing file
will be made.

The trackside signals are controlled using 74HC595 shift registers. Since each
signal lamp utilizes a single red/green LED, internally wired back-to-back, two
shift register bits are needed for each lamp to obtain the desired four state
indications; off, red, green, and yellow. This is similar to the previous Basic
Stamp design. The grade crossing signal lamps are also controlled by this shift
register.

The block detector, sensor, and keypad inputs are interfaced using I2C 32 Channel
Pi expansion boards. These boards use the Microchip MCP23017. The keypads are
used for turnout positioning input.

There is copious documentation contained in the program code which explains the
design and operation in greater detail. All programs can be viewed in a text
editor or the program listing binder.

USAGE:
   $ExecutableName [-h] [-q] [-f] [-i] [-d <lvl>] [-c] [-o|-m|-c <num>]
          [-s [r]<range>] [-t [r]<range>] [-b <range>] [-g 1|2] [-k] [-n]
          [-p] [-r] [-v <num>] [-x] [-y] [-z] [-a] [-u Tx[p]:t1,t2,...]
          [-w]

   -h            Show program help.

   -q            Runs the program in quiet mode. Suppresses all console
                 messages. Useful when running the program using autostart.

   -d <lvl>      Run at specified debug level; 0-3. Higher level increases
                 message verbosity. Uncomment Forks::Super::DEBUG statement
                 in main code to see Forks related debug output. Note that
                 level 3 causes output of child process messages. This may
                 result in a flood of message output until DnB.pl is ctrl+c
                 terminated and then restarted with a lower debug level.

   -i            Detect and display the I2C addresses; runs i2cdetect in
                 the background. Expected active addresses are:

                    Block detectors: 0x20
                    Track sensors:   0x21
                    Yard keypad:     0x22
                    Button input:    0x23
                    Turnouts 1-16:   0x41
                    Turnouts 17-32:  0x42
                    Not Used:        0x70

   -y            Send console output to the serial port device.
                 Device: $SerialDev   Baud: $SerialBaud

```

```
    -f           Backup existing TurnoutDataFile.txt file, if any, and
                 create a new file with default values. The program exits
                 once the file is created.

    -x           Disable shutdown button check during power on. Used for
                 testing when button hardware is not physically connected.

    -z           Enable toggle of GPIO20_TEST pin. Used to view main loop
                 timing on a scope. Each code section toggles the GPIO
                 state.
                    A - Top of loop          G - Process Signals
                    B - Read sensors         H - Process Yard Route
                    C - Process holdover     I - Read keypad
                    D - Process midway       J - MidwayTrack
                    E - Process wye          K - WyeTrack
                    F - Grade Crossing (2)   L - Shutdown button

    -o|m|c <num> Set the specified servo to its open, middle, or closed
                 position. Used for servo mechanical adjustments. Program
                 exits once position is set. <num> = 0 sets all servos to
                 the specified position.

    -b <range>   Run sensor bit test. <range> specifies the chip numbers
                 to use, 1 thru 4. e.g. 1 (chip 1), 1,2 (chips 1 and 2),
                 1:4 (chips 1 thru 4). The associated sensor bits are read
                 and displayed. This test runs until terminated by ctrl+c.

    -g 1|2       Run grade crossing test using the specified crossing,
                 1, 2, or both (comma separated). The grade crossing lamps
                 are flashed and gates raised and lowered. This test runs
                 until terminated by ctrl+c.

    -k           Run the keypad test; pressed buttons will be displayed.
                 The 1st entry LED will toggle for each 4x4 keypad button
                 press. Single/double button presses on the 1x4 keypads
                 will also be displayed. This test runs until it is
                 terminated by ctrl+c.

    -n           Run sensor tone test; all sensors are included. An ID
                 number of tones sound when a sensor becomes active and
                 a double tone sounds when the sensor becomes inactive.
                 This facilitates sensor operability testing at remote
                 layout locations; e.g. by manually blocking an IR light
                 path. This test runs until terminated by ctrl+c.

    -p           Run the sound player test. Used to select and audition
                 the available sound files. This test runs until it is
                 terminated.

    -r <range>   Run the power polarity relay test. <range> specifies the
                 relay to test; 1, 2, or 3. Specify 0 to test all relays.
                 The relay is energized for 5 seconds and de-energized for
                 5 seconds. Test runs until it is terminated by ctrl+c.

    -s <range>   Run signal test. <range> specifies the signal numbers to
                 use, 1 thru 12. e.g. 1 (signal 1), 1,5 (signals 1 and 5),
                 1:5 (signals 1 thru 5). Preface with 'r' (r1:5) to test
                 the specified signals in random instead of sequential
                 order. <range> specified as Red, Grn, Yel, or Off will
                 set all signals to the specified condition. <range>
```

```
                specified as color:nmbr will set the specified signal to
                the specified color. Preface with 'g' to include grade
                crossings 1 and 2. This test runs until terminated by
                ctrl+c.

   -t <range>   Run turnout test. <range> specifies the turnout numbers
                to use, 1 thru 29. e.g. 1 (turnout 1), 1,5 (turnouts 1
                and 5), 1:5 (turnouts 1 thru 5). Preface with 'r' (r1:5)
                to test the specified turnouts randomly instead of sequen-
                tial order. Add 'w' (w1:5, wr1:5) to wait for the opera-
                tion to complete before staring another. <range> specified
                Open, Middle, or Close will set all turnoutss to the
                specified position. <range> specified as position:nmbr
                will set the specified turnout to the specified position.
                This test runs until terminated by ctrl+c.

   -u <param>   Run the servo temperature adjust test. <param> specifies
                a servo number and one or more temperatures in degrees C.
                The first temperature is set and the servo is positioned.
                The cycle repeats for each specified temperature. Each
                position is tested unless a single position is specified;
                o, m, or c.

                A low tone is sounded at the start of each position. A high
                tone sounds for each temperature change. Changes occur at 1
                second intervals. This test runs until terminated by ctrl+c.

   -v <num>     Sets the sound volume to the specified percentage value;
                1-100. Default ${AudioVolume}% is used when not specified.

   -a           Simulation mode. This test simulates train movements and
                turnout operations on the layout. The default 'EndToEnd'
                simulation runs until terminated by ctrl+c. Sensorbits,
                yard routes, and turnout positions that are stored in the
                %SimulationData hash are used instead of the actual layout
                input. In this mode, the operational code is exercised
                without actually running a train on the layout. Refer to
                the %SimulationData hash for details. Use debug level 0 to
                display additional simulation data on the console.

   -w           Webserver enable. This option specifies that the webserver
                interface should be enabled. When active, an external web
                browser can connect to the Rpi and view various operational
                data in near real time. The currently configured connection
                point is: DnB-Model-RR:$ListenPort .

==============================================================================

));

# ===========================================================================
# MAIN PROGRAM
# ===========================================================================

# Process user specified CLI options.
getopts('haqipfxzknywb:d:t:s:o:m:c:g:v:r:u:', \%Opt);

if (defined($Opt{d})) {
   if ($Opt{d} =~ m/^\d+$/ and $Opt{d} >= 0 and $Opt{d} <= 3) {
      $DebugLevel = $Opt{d} + 0;
```

```
#      $Forks::Super::DEBUG = 1;   # Uncomment to see Forks::Super debug output.
   }
   else {
      &DisplayError("main, Invalid DebugLevel specified: $Opt{d}");
      exit(1);
   }
}

# ---------------------------------------------------------------------------
# Display help text if requested.
#
if (defined($Opt{h})) {
   print $UsageText;
   exit(0);
}

# ---------------------------------------------------------------------------
# Display I2C addresses if requested.
#
if (defined($Opt{i})) {
   print "\nActive I2C addresses:\n\n";
   system("sudo i2cdetect -y 1");
   print "\n";
   exit(0);
}

# ---------------------------------------------------------------------------
# Create new TurnoutDataFile if requested.
#
if (defined($Opt{f})) {
   if (-e $TurnoutFile) {
      my $backupFile = $TurnoutFile;
      $backupFile =~ s/txt$/bak/;
      my @Array = ();
      exit(1) if (&ReadFile($TurnoutFile, \@Array, "NoTrim"));
      foreach my $rec (@Array) {
         chomp($rec);
      }
      exit(1) if (&WriteFile($backupFile, \@Array, ""));
      unless (-e $backupFile) {
         &DisplayError("main, Failed to create backup file $backupFile");
         exit(1);
      }
   }
   if (&ProcessTurnoutFile($TurnoutFile, "Write", \%TurnoutData)) {
      &DisplayError("main, Failed to create $TurnoutFile");
      exit(1);
   }
   if (-e $TurnoutFile) {
      &DisplayMessage("Default TurnoutDataFile successfully created.");
   }
   exit(0);
}

# ---------------------------------------------------------------------------
# Setup for processing keyboard entered signals.
#
foreach my $sig ('INT','QUIT','TERM') {     # Catch termination signals
   $SIG{$sig} = \&Ctrl_C;
}
```

```perl

# ---------------------------------------------------------------------------
# Configure for buffer autoflush.
#
select (STDERR);
$| = 1;
select (STDOUT);
$| = 1;

# ---------------------------------------------------------------------------
# Kill orphan child processes and parent/child intercommunication files,
# if any. This will occur if the program abnormally terminates.
#
my @list = `ps -ef | grep DnB.pl`;
foreach my $line (@list) {
   if ($line =~ m/^\w+\s+(\d+)\s+1\s/) {
      system("kill -9 $1");
   }
}
my $result = `rm -rf /dev/shm/.fh*`;

# ---------------------------------------------------------------------------
# Open the serial port if specified.
#
if (defined($Opt{y})) {
   if (&OpenSerialPort(\$SerialPort, $SerialDev, $SerialBaud)) {
      &DisplayWarning("main, Failed to open serial port. $SerialDev");
   }
   unless (defined($Opt{q})) {
      print STDOUT "$$ Serial port $SerialDev open, $SerialBaud baud.\n";
   }
}

# ===========================================================================
# Tell the world we're up and running.
#
&DisplayMessage("=== DnB program start ===");
$MainRun = 1;

# ---------------------------------------------------------------------------
# Set audio volume if specified.
#
if (defined($Opt{v})) {
   my($vol) = $Opt{v} =~ m/^(\d+)/;
   if ($vol ne '' and $vol > 0 and $vol <= 100) {
      $AudioVolume = "$vol";
   }
   else {
      &DisplayError("main, Invalid sound volume specified: $Opt{v}");
      exit(1);
   }
}

# ---------------------------------------------------------------------------
# Initialize the GPIO pins associated with the Signal LED Driver. Check the
# shutdown button (0 if pressed). If pressed, terminate this program but
# don't shutdown Linux OS.
#
if (&Init_SignalDriver(\%GpioData, scalar(keys %SignalData)*2)) {
   exit(1);
```

```
}
else {
   # Check for user press of shutdonw button to abort startup. Skip check if
   # -x option or any test option is specified.
   unless (defined($Opt{x}) or defined($Opt{p}) or defined($Opt{k}) or
           defined($Opt{g}) or defined($Opt{b}) or defined($Opt{t}) or
           defined($Opt{s}) or defined($Opt{o}) or defined($Opt{m}) or
           defined($Opt{c}) or defined($Opt{n}) or defined($Opt{r}) or
           defined($Opt{a})) {
      my $buttonPress = $GpioData{'GPIO21_SHDN'}{'Obj'}->read;
      if ($buttonPress == 0) {
         print "$$ main, Shutdown button pressed. Aborting DnB startup.\n";
         print "$$ main, Specify -x option to bypass this check.\n\n";
         &PlaySound("Unlock.wav");
         sleep 1;
         exit(0);
      }
      &PlaySound("G.wav");
   }
}

# ---------------------------------------------------------------------------
# Initialize the I2C MCP23017 sensor chips on the I/O PI Plus board.
#
for (my $chip = 1; $chip <= scalar keys(%SensorChip); $chip++) {
   if ($SensorChip{$chip} == 0) {
      &DisplayDebug(1, "main, Skip chip $chip I2C_Address 0, code debug.");
      next;
   }
   &DisplayMessage("Initializing sensor I2C MCP23017 $chip ...");
   exit(1) if (&I2C_InitSensorDriver($chip, \%MCP23017, \%SensorChip));
}

# ---------------------------------------------------------------------------
# Start the child processes.
foreach my $indx (sort keys (%ChildProcess)) {
   next if ($ChildProcess{$indx}{'Name'} eq 'WebserverChild' and not
            defined($Opt{w}));
   my($pid) = fork $ChildProcess{$indx}{'Code'}, $ChildProcess{$indx}{'Opt'};
   if (!defined($pid)) {
      &DisplayError("main, Failed to start $ChildProcess{$indx}{'Name'}. $!");
      exit(1);
   }
   else {
      $ChildProcess{$indx}{'Pid'} = $pid;

      # Save needed SignalChild pid in each %GradeCrossingData entry.
      if ($ChildProcess{$indx}{'Name'} =~ m/SignalChild/) {
         foreach my $gc (sort keys (%GradeCrossingData)) {
            $GradeCrossingData{$gc}{'SigPid'} = $pid;
         }
      }

      # Need a copy of the grade crossing PID's in the %GradeCrossingData hash.
      if ($ChildProcess{$indx}{'Name'} =~ m/^GcChild\s*(\d+)/) {
         $GradeCrossingData{$1}{'Pid'} = $pid;
      }
      &DisplayDebug(1, "main, $ChildProcess{$indx}{'Name'}: $pid");
   }
}
```

```perl

# ---------------------------------------------------------------------------
# Load the data from the turnout last position file into the %TurnoutData
# hash.
#
&DisplayMessage("Reading turnout last position file ...");
&ProcessTurnoutFile($TurnoutFile, "Read", \%TurnoutData);

# ---------------------------------------------------------------------------
# Initialize the I2C servo driver boards to the PWM position specified in
# %TurnoutData for each servo. Exit if positioning servo(s) for mechanical
# adjustment of turnout points (-o, -m, or -c options).
#
if (defined($Opt{o})) {
   exit(&InitTurnouts(\%ServoBoardAddress, \%TurnoutData, $Opt{o}, 'Open'));
}
elsif (defined($Opt{m})) {
   exit(&InitTurnouts(\%ServoBoardAddress, \%TurnoutData, $Opt{m}, 'Middle'));
}
elsif (defined($Opt{c})) {
   exit(&InitTurnouts(\%ServoBoardAddress, \%TurnoutData, $Opt{c}, 'Close'));
}
else {
   if (&InitTurnouts(\%ServoBoardAddress, \%TurnoutData, '', '')) {
      &PlaySound("CA.wav");
      sleep 1;
      exit(1);
   }
}

# ---------------------------------------------------------------------------
# Get the initial ambient temperature value and store for use when positioning
# the gates and semaphore. The GetTemperature subroutine is in Turnout.pm. The
# subroutine also creates %TurnoutData{'00'}{'Timeout'} to indicate the next
# update time.
if (&GetTemperature(\%TurnoutData) == 0) {
   &DisplayWarning("main, GetTemperature did not return a value.");
}
else {
   my $tempF = ($TurnoutData{'00'}{'Temperature'} * (9/5)) + 32;
   &DisplayMessage("Ambient temperature is: $TurnoutData{'00'}{'Temperature'} " .
                   "C  (" . sprintf("%.1f F)", $tempF));
}

# ===========================================================================
# Perfom CLI specified testing.

# -----
# Run TestSensorBits in DnB_Sensor.pm if specified.
# -----
if (defined($Opt{b})) {
   exit(&TestSensorBits($Opt{b}, \%MCP23017, \%SensorChip, \%SensorState));
}

# -----
# Run TestSensorTones in DnB_Sensor.pm if specified.
# -----
if (defined($Opt{n})) {
   exit(&TestSensorTones(\%MCP23017, \%SensorChip, \%SensorState, \%SensorBit));
}
```

```
# -----
# Run TestKeypad in DnB_Sensor.pm if specified.
# -----
if (defined($Opt{k})) {
   exit(&TestKeypad('1', \%KeypadData, \%ButtonData, \%GpioData, \%MCP23017,
                      \%SensorChip, \$ChildProcess{'02'}{'Pid'},
                      \$ChildProcess{'03'}{'Pid'}));
}

# -----
# Run TestGradeCrossing in DnB_GradeCrossing.pm if specified.
# -----
if (defined($Opt{g})) {
   sleep 0.5;               # Delay for GcChildProcess message output.
   exit(&TestGradeCrossing($Opt{g}, \%GradeCrossingData, \%TurnoutData));
}

# -----
# Run TestSignals in DnB_Signal.pm if specified. Options for signal testing can
# include grade crossing and gate (turnout code) testing.
# -----
if (defined($Opt{s})) {
   sleep 0.5;               # Delay for SignalChildProcess message.
   exit(&TestSignals($Opt{s}, $ChildProcess{'01'}{'Pid'}, \%SignalData,
                  \%GradeCrossingData, \%SemaphoreData, \%TurnoutData));
}

# -----
# Run TestTurnouts in DnB_Turnout.pm if specified.
# -----
if (defined($Opt{t})) {
   exit(&TestTurnouts($Opt{t}, \%TurnoutData));
}

# -----
# Run TestServoAdjust in DnB_Turnout.pm if specified.
# -----
if (defined($Opt{u})) {
   exit(&TestServoAdjust($Opt{u}, \%TurnoutData));
}

# -----
# Run TestSound in DnB_Yard.pm if specified.
# -----
if (defined($Opt{p})) {
   my $soundFileDir = substr($SoundPlayer, rindex($SoundPlayer, " ")+1);
   exit(&TestSound($soundFileDir));
}

# -----
# Run TestRelay in DnB_Yard.pm if specified.
# -----
if (defined($Opt{r})) {
   exit(&TestRelay($Opt{r}, \%GpioData));
}

# ==========================================================================
# Start main program loop.
#
```

```
if (defined($Opt{a})) {
   &DisplayMessage("--> DnB SIMULATION MODE start <--");
   exit(1) if (&InitSimulation('EndToEnd', \%SimulationData));
   $MainRun = 2;
}
else {
   &DisplayMessage("=== DnB main loop start ===");
   $MainRun = 3;                    # Ctrl+c updates TurnoutData.txt file.
}

my ($webserverUpdate) = 0;   # Webserver update control variable.

while ($MainRun) {

# Clear accumulator variables for webserver data.
   my($sensorWork, $signalWork) = ('','');

# -----
# Read the sensors and store values in %SensorState hash. If running in
# simulation mode (-a), use simulated sensor values.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(1) if (defined($Opt{z}));  # A
   if (defined($Opt{a})) {
      &SimulationStep(\%SensorBit, \$SensorState{'1'}, \$SensorState{'2'},
                      \%SimulationData, \%TurnoutData, \%YardRouteData);
   }
   else {
      &DisplayDebug(2, "main - Driver: $SensorChip{'1'}{'Obj'}");

      $SensorState{'1'} =
         ($SensorChip{'1'}{'Obj'}->read_byte($MCP23017{'GPIOB'}) << 8) |
          $SensorChip{'1'}{'Obj'}->read_byte($MCP23017{'GPIOA'});
      $SensorState{'2'} =
         ($SensorChip{'2'}{'Obj'}->read_byte($MCP23017{'GPIOB'}) << 8) |
          $SensorChip{'2'}{'Obj'}->read_byte($MCP23017{'GPIOA'});

      if (defined($Opt{w})) {   # webserver data
         $sensorWork = (($SensorState{'2'} << 16) | $SensorState{'1'});
      }
   }

# -----
# Set the sensor activated turnouts and polarity relays.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(0) if (defined($Opt{z}));  # B
   &ProcessHoldover(\%TrackData, \%SensorBit, \%SensorState,
                    \%TurnoutData, \%GpioData);

   $GpioData{'GPIO20_TEST'}{'Obj'}->write(1) if (defined($Opt{z}));  # C
   &ProcessMidway(\%TrackData, \%SensorBit, \%SensorState,
                  \%TurnoutData);

   $GpioData{'GPIO20_TEST'}{'Obj'}->write(0) if (defined($Opt{z}));  # D
   &ProcessWye(\%TrackData, \%SensorBit, \%SensorState,
               \%TurnoutData, \%GpioData);

# -----
# Call ProcessGradeCrossing to check and process the grade crossing sensors.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(1) if (defined($Opt{z}));  # E
```

```
   foreach my $gc (sort keys(%GradeCrossingData)) {
      next if ($gc eq '00');
      &ProcessGradeCrossing($gc, \%GradeCrossingData, \%SensorBit,
                   \%TurnoutData, \%MCP23017, \%SensorState, $WebDataDir);
      # last; # uncomment for one signal debug
   }

# -----
# Set track signals using the block detector sensor bits.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(0) if (defined($Opt{z}));  # F
   my %signalWork = ();                        # Initialize working hash.
   my @activeList = ();                        # Active block list for -w.
   my $signalStr = '';                         # Signal list for -w.
   my %sigLiveColor = ();                      # Signal list for lamp color -w.
   foreach my $color ('Grn','Yel','Red') {
      foreach my $block ('00','01','02','03','04','05','06','07','08','09') {
         my $sensorBits = $SensorState{ $SensorBit{$block}{'Chip'} };
         my $bitMask = 1;
         if ($SensorBit{$block}{'Bit'} =~ m/(GPIO.)(\d)/) {
            $bitMask = $bitMask << 8 if ($1 eq 'GPIOB');
            $bitMask = $bitMask << $2;
            &DisplayDebug(3, "main, color: $color   block: $block" .
                          "   sensorBits: " . sprintf("%0.16b", $sensorBits) .
                          "   bitMask: " . sprintf("%0.16b", $bitMask));
         }
         if ($sensorBits & $bitMask) {   # Block active if not zero

            # Available color settings?
            if (exists $SignalColor{$block}{$color}) {
               my @sigColorList = split(",", $SignalColor{$block}{$color});
               &DisplayDebug(2, "main, block: $block   color: " .
                             "$color   sigColorList: @sigColorList");
               foreach my $signal (@sigColorList) {
                  $signalWork{$signal} = $color;
               }
            }

            # Add to active block list for live web page file selection.
            if ($color =~ m/Red/i) {        # Process only during last color.
               my $bNum = $block +1;
               $bNum = "0$bNum" if (length($bNum) == 1);
               push (@activeList, join('', 'B', $bNum));
            }
         }
      }
   }

   # Activate the new signal values.
   for my $signal ('01','02','03','04','05','06','07','08','09','10','11','12') {
      my $color = 'Off';
      $color = $signalWork{$signal} if (exists ($signalWork{$signal}));

      if (defined($Opt{w})) {   # webserver data
         $signalStr = join(',', $signalStr, join('=', "L${signal}", $color));
         $sigLiveColor{$signal} = $color;
      }

      # Skip if signal is already at the proper color.
      next if ($SignalData{$signal}{'Current'} eq $color);
```

```

      # Set new signal color.
      if (exists ($SemaphoreData{$signal})) {
         if (&SetSemaphoreSignal($signal, $color, $ChildProcess{'01'}{'Pid'},
                          \%SignalData, \%SemaphoreData, \%TurnoutData)) {
            &DisplayError("main, SetSemaphoreSignal $signal " .
                          "'$color' returned error.");
         }
      }
      else {
         if (&SetSignalColor($signal, $color, $ChildProcess{'01'}{'Pid'},
                             \%SignalData, '')) {
            &DisplayError("main, SetSignalColor $signal " .
                          "'$color' returned error.");
         }
      }
   }

# -----
# Process inprogress turnout route setting.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(1) if (defined($Opt{z}));  # G
   &YardRoute(\%YardRouteData, \%TurnoutData);

# -----
# Get and process yard route input from user.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(0) if (defined($Opt{z}));  # H
   &GetYardRoute(\%YardRouteData, \%KeypadData, \%GpioData,
                 $ChildProcess{'02'}{'Pid'});

# -----
# Process user single button input.
# -----
   my $buttonInput = Forks::Super::read_stderr($ChildProcess{'03'}{'Pid'});
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(1) if (defined($Opt{z}));  # I
   &HoldoverTrack($buttonInput, \%TurnoutData, \%TrackData, \%GpioData);

   $GpioData{'GPIO20_TEST'}{'Obj'}->write(0) if (defined($Opt{z}));  # J
   &MidwayTrack($buttonInput, \%ButtonData, \%TurnoutData, \%TrackData,
                \%SensorBit, \%SensorState);

   $GpioData{'GPIO20_TEST'}{'Obj'}->write(1) if (defined($Opt{z}));  # K
   &WyeTrack($buttonInput, \%ButtonData, \%TurnoutData, \%TrackData,
             \%SensorBit, \%SensorState, \%GpioData);

# -----
# Update the ambient temperature value. A new timeout is set as part of
# the call to this subroutine. See code in Turnout.pm.
# -----
   &GetTemperature(\%TurnoutData) if ($TurnoutData{'00'}{'Timeout'} < time);

# -----
# Collect and save data for webserver. Sensor and signal data was collected
# above. Need to do the turnout data here. When the $webserverUpdate control
# variable is zero, update and then reset its value.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(0) if (defined($Opt{z}));  # L
   if (defined($Opt{w}) and $webserverUpdate-- <= 0) {
      my(@data) = ("Sensor: $sensorWork");
```

```
      $signalStr =~ s/^,//;
      push(@data, "Signal: $signalStr");
      foreach my $turnout (sort keys(%TurnoutData)) {
         next if ($turnout eq '00');
         my($values) = '';
         foreach my $attr ('Pos','Rate','Open','Middle','Close','MinPos',
                           'MaxPos','Id') {
            $values = join(':', $values, $TurnoutData{$turnout}{$attr});
         }
         $values =~ s/^://;
         push(@data, join('=', "T${turnout}", $values));
      }
      &WriteFile("$WebDataDir/sensor.dat", \@data, '');

      # Store the appropriate overlay file names for the mainline live data
      # page. The @activeList array holds the active track blocks that was
      # built by the above track signal code.
      my ($hFile, $mFile, $yFile) = ('', '', '');
      foreach my $block (@activeList) {
         if ($block ge 'B01' and $block le 'B03') {
            $hFile = join('', $hFile, $block);
         }
         if ($block ge 'B03' and $block le 'B06') {
            $mFile = join('', $mFile, $block);
         }
         if ($block ge 'B06' and $block le 'B10') {
            $yFile = join('', $yFile, $block);
         }
      }
      my(@array) = (join('', 'DnB-H-', $hFile, '.png'));
      &WriteFile("$WebDataDir/h-overlay.dat", \@array, '');
      @array = (join('', 'DnB-M-', $mFile, '.png'));
      &WriteFile("$WebDataDir/m-overlay.dat", \@array, '');
      @array = (join('', 'DnB-Y-', $yFile, '.png'));
      &WriteFile("$WebDataDir/y-overlay.dat", \@array, '');

      # Store the appropriate signal color overlay file names for the mainline
      # live data page. %sigLiveColor holds the current signal colors.
      foreach my $signal (sort keys(%sigLiveColor)) {
         my $sig = join('', 'L', $signal);
         @array = (join('', 'DnB-', $sig, '-', $sigLiveColor{$signal}, '.png'));
         &WriteFile("$WebDataDir/$sig-overlay.dat", \@array, '');
      }

      # Update the yard route overlay file. The @data array holds the
      # current position data that was built above. Called code is located
      # in Yard.pm.
      &YardLiveOverlay(\@data, $WebDataDir);

      $webserverUpdate = 10;
   }

# -----
# Initiate shutdown if requested by the user. ShutdownRequest will return 1
# if the shutdown button has been pressed and not aborted with another press
# within 5 seconds.
#
# Despite eventual RPi shutdown, the last state of the hardware will
# continue to drive the associated circuitry as long as power is on.
# The following orderly shutdown ensures all servos, LEDs, relays, and
```

```
# sound modules are set to off.
# -----
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(1) if (defined($Opt{z}));  # M
   $Shutdown = &ShutdownRequest('FF', \%ButtonData, \%GpioData);
   $GpioData{'GPIO20_TEST'}{'Obj'}->write(0) if (defined($Opt{z}));  # N
   sleep 0.090;        # Delay before next main loop iteration
   last if ($Shutdown == 1);
}

# Perform orderly shutdown; button or Ctrl+C initiated.
&DisplayMessage("=== DnB program shutting down ===");

if ($Shutdown == 1) {  # Ctrl+C terminates child processed.
   &DisplayMessage("Stop child processes.");
   foreach my $indx (sort keys %ChildProcess) {
      system("kill -9 $ChildProcess{$indx}{'Pid'}");
   }
}

&DisplayMessage("Raise crossing gates and semaphores.");
foreach my $turnout (sort keys(%TurnoutData)) {
   if ($TurnoutData{$turnout}{'Id'} =~ m/semaphore/i or
       $TurnoutData{$turnout}{'Id'} =~ m/gate/i) {
      &MoveTurnout('Open', $turnout, \%TurnoutData);
   }
}

&DisplayMessage("Wait for turnout moves to complete.");
my $moveWait = 6;
while ($moveWait > 0) {
   my @inprogress = ();
   foreach my $turnout (sort keys(%TurnoutData)) {
      if ($TurnoutData{$turnout}{'Pid'} != 0) {
         push (@inprogress, $turnout);
      }
   }
   last if ($#inprogress < 0);
   &DisplayMessage("   Inprogress: " . join(' ', @inprogress));
   sleep 1;                      # Wait 1 second.
   $moveWait--;
}

&DisplayMessage("Turn off all servo channels.");
foreach my $key (sort keys(%ServoBoardAddress)) {
   my $I2C_Address = $ServoBoardAddress{$key};
   my $driver = RPi::I2C->new($I2C_Address);
   unless ($driver->check_device($I2C_Address)) {
      &DisplayError("Failed to instantiate I2C address: " .
                     sprintf("0x%.2x",$I2C_Address));
      next;
   }

   my(%PCA9685) = ('ModeReg1' => 0x00, 'ModeReg2' => 0x01,
                   'AllLedOffH' => 0xFD, 'PreScale' => 0xFE);
   $driver->write_byte(0x10, $PCA9685{'AllLedOffH'});  # Orderly shutdown.
   undef($driver);
}

&DisplayMessage("Turn off all signal LEDs.");
$GpioData{'GPIO22_DATA'}{'Obj'}->write(0);
```

```
for my $pos (reverse(0..31)) {
   $GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);       # Set SCLK low.
   $GpioData{'GPIO27_SCLK'}{'Obj'}->write(1);       # Set SCLK high
}
$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);          # Set SCLK low.
$GpioData{'GPIO17_XLAT'}{'Obj'}->write(1);          # Set XLAT high
$GpioData{'GPIO17_XLAT'}{'Obj'}->write(0);          # Set XLAT low.

&DisplayMessage("Turn off GPIO driven relays and indicators");
foreach my $gpio (sort keys(%GpioData)) {
   if ($GpioData{$gpio}{'Desc'} =~ m/Polarity relay/i or
       $GpioData{$gpio}{'Desc'} =~ m/first entry/i or
       $GpioData{$gpio}{'Desc'} =~ m/route lock/i) {
      $GpioData{$gpio}{'Obj'}->write(0);
   }
}

# Turn off holdover position LEDs and silence sound modules.
$SensorChip{'4'}{'Obj'}->write_byte(0, $MCP23017{'OLATB'});

# Save current turnout data to file.
&ProcessTurnoutFile($TurnoutFile, "Write", \%TurnoutData);
&DisplayMessage("Turnout position data saved.");
sleep 1;
&DisplayMessage("=== DnB program termination ===");

system("sudo shutdown -h now") if ($Shutdown == 1);
exit(0);
```