```
# ============================================================================
# FILE: DnB_Message.pm                                              7/06/2020
#
# SERVICES:  DnB MESSAGE AND UTILITY FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides message and utility functions used by the DnB
#    model railroad control program. A number of these functions are present
#    for possible future use.
#
# PERL VERSION: 5.24.1
#
# ============================================================================
use strict;
# ----------------------------------------------------------------------------
# Package Declaration
# ----------------------------------------------------------------------------
package DnB_Message;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   OpenSerialPort
   ShutdownRequest
   PlaySound
   Ctrl_C
   ReadFile
   ReadBin
   ReadFileHandle
   WriteFile
   WriteFileAppend
   DisplayMessage
   DisplayError
   DisplayWarning
   DisplayDebug
   Trim
   TrimArray
   SplitIt
   HexToAscii
   DateTime
   DelDirTree
   GrepFile
   ShuffleArray
);

use Time::HiRes qw(gettimeofday sleep);

# ============================================================================
# FUNCTION:  OpenSerialPort
#
# DESCRIPTION:
#    This routine opens the Raspberry serial port using the specified device
#    and baud rate and returns the object to the caller. The serial port is
#    used to communicate message information to a monitoring terminal.
#
# CALLING SYNTAX:
#    $result = &OpenSerialPort(\$SerialObj, $Device, $Baud);
#
# ARGUMENTS:
#    $SerialObj      Pointer to serial object variable
```

```
#    $Device         Serial device to associated to object
#    $Baud           Communication baud rate.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#    $SerialObj = Set to object reference
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub OpenSerialPort {

   my($SerialObj, $Device, $Baud) = @_;

   &DisplayDebug(2, "OpenSerialPort, Device: $Device   Baud: $Baud");
   undef($$SerialObj);

   $$SerialObj = RPi::Serial->new($Device, $Baud);
   unless ($$SerialObj) {
      &DisplayError("OpenSerialPort, Serial device not accessable: $Device");
      return 1;
   }
   return 0;
}

# ============================================================================
# FUNCTION:  ShutdownRequest
#
# DESCRIPTION:
#    This routine is called to check and process a user requested shutdown. This
#    state sequence uses a dedicated shutdown button and is called as part of
#    main program loop. Once initiated, another button press during timeout will
#    abort the shutdown. The shutdown button reads 0 when pressed and 1 when
#    released due to GPIO21 configured with pullup.
#
# CALLING SYNTAX:
#    $result = &ShutdownRequest($Button, \%ButtonData, \%GpioData);
#
# ARGUMENTS:
#    $Button               Button index in %ButtonData hash.
#    $ButtonData           Pointer to %ButtonData hash.
#    $GpioData             Pointer to %GpioData hash.
#
# RETURNED VALUES:
#    0 = Run,  1 = Shutdown.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ShutdownRequest {
   my($Button, $ButtonData, $GpioData) = @_;
   my($buttonPress, @tones, $tone);

   $buttonPress = $$GpioData{ $$ButtonData{$Button}{'Gpio'} }{'Obj'}->read;

   # State 2
   if ($$ButtonData{$Button}{'Wait'} == 1) {  # Waiting for button release?
      if ($buttonPress == 1) {                    # Is button now released?
         $$ButtonData{$Button}{'Wait'} = 0;
         $$ButtonData{$Button}{'Shutdown'} = 1;    # Start shutdown timeout
```

```
121            &DisplayMessage("ShutdownRequest, RPi shutdown initiated. " .
122                            "Press button again to abort.");
123         }
124      }
125
126      # State 4
127      elsif ($$ButtonData{$Button}{'Wait'} == 2) {    # Waiting final release?
128         if ($buttonPress == 1) {                     # Is button now released?
129            $$ButtonData{$Button}{'Wait'} = 0;
130            &DisplayMessage("ShutdownRequest, RPi shutdown aborted.");
131            sleep 0.1                         ;       # Button debounce.
132         }
133      }
134
135      # State 1 and 3
136      elsif ($buttonPress == 0) {                       # Is button pressed?
137         if ($$ButtonData{$Button}{'Shutdown'} == 1) {  # Timeout inprogress?
138            $$ButtonData{$Button}{'Shutdown'} = 0;      # Abort shutdown.
139            $$ButtonData{$Button}{'Step'} = 0;          # Reset step position.
140            $$ButtonData{$Button}{'Wait'} = 2;       # Wait for button release.
141            &PlaySound("Unlock.wav");
142         }
143         else {
144            $$ButtonData{$Button}{'Wait'} = 1;       # Wait for button release.
145         }
146      }
147
148      # State 3
149      elsif ($$ButtonData{$Button}{'Shutdown'} == 1) {   # Timeout inprogress?
150         if (gettimeofday > $$ButtonData{$Button}{'Time'}) {
151            $$ButtonData{$Button}{'Time'} = gettimeofday + 1;
152            @tones = split(",", $$ButtonData{$Button}{'Tones'});
153            $tone = $tones[$$ButtonData{$Button}{'Step'}++];
154            &PlaySound("${tone}.wav");
155            if ($$ButtonData{$Button}{'Step'} > $#tones) {
156               sleep 2;                                 # Time for last tone.
157               $$ButtonData{$Button}{'Time'} = 0;       # Reset for testing.
158               $$ButtonData{$Button}{'Shutdown'} = 0;
159               $$ButtonData{$Button}{'Step'} = 0;
160               return 1;                                          # Shutdown
161            }
162         }
163      }
164      return 0;
165   }
166
167   # ============================================================================
168   # FUNCTION:  PlaySound
169   #
170   # DESCRIPTION:
171   #    This routine plays the specified sound file using the player application
172   #    defined by global variable $main::SoundPlayer. Sound file playback is done
173   #    asynchronously without waiting for playback to complete.
174   #
175   # CALLING SYNTAX:
176   #    $result = &PlaySound($SoundFile, $Volume);
177   #
178   # ARGUMENTS:
179   #    $SoundFile          File to be played.
180   #    $Volume             Optional; volume level.
```

```
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::SoundPlayer, $main::AudioVolume
# ============================================================================
sub PlaySound {
   my($SoundFile, $Volume) = @_;
   my($vol);
   my($filePath) = substr($main::SoundPlayer, rindex($main::SoundPlayer, " ")+1);

   &DisplayDebug(2, "PlaySound, entry.  filePath: $filePath  SoundFile: $SoundFile");

   if (-e "${filePath}/${SoundFile}") {
      if ($Volume =~ m/^(\d+)/) {
         $vol = $1;
      }
      else {
         $vol = $main::AudioVolume;
      }
      system("/usr/bin/amixer set PCM ${vol}% >/dev/null");
      system("${main::SoundPlayer}/$SoundFile &");
   }
   else {
      &DisplayError("PlaySound, Sound file not found: ${filePath}/${SoundFile}");
      return 1;
   }
   return 0;
}

# ============================================================================
# FUNCTION:  Ctrl_C
#
# DESCRIPTION:
#    This routine is used to handle console entered ctrl+c input. When entered,
#    the INT signal is sent to all child processes. Each child process will run
#    this routine in their forked context and terminate. The ChildName variable,
#    set by each child process when it starts, serves to identify the exiting
#    child process.
#
#    The main program performs an orderly shutdown of the turnout servo driver
#    boards to prevent lockups that require a power cycle to correct. It then
#    saves the current turnout position data if running at operations level,
#    $main:: MainRun == 2.
#
# CALLING SYNTAX:
#    None.
#
# ARGUMENTS:
#    None.
#
# RETURNED VALUES:
#    None.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::MainRun, $main::TurnoutFile, %main::TurnoutData, $main::ChildName,
#    $main::$opt{q}, %main::ServoBoardAddress

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

```
sub Ctrl_C {
   my($driver, $I2C_Address);
   my(%PCA9685) = ('ModeReg1' => 0x00, 'ModeReg2' => 0x01, 'AllLedOffH' => 0xFD,
                   'PreScale' => 0xFE);

   undef ($main::Opt{q});                        # Ensure console messages are on.
   if ($main::ChildName eq 'Main') {
      foreach my $key (sort keys(%main::ServoBoardAddress)) {
         $I2C_Address = $main::ServoBoardAddress{$key};
         $driver = RPi::I2C->new($I2C_Address);
         unless ($driver->check_device($I2C_Address)) {
            &DisplayError("Ctrl_C, Failed to instantiate I2C address: " .
                          sprintf("0x%.2x",$I2C_Address));
            next;
         }
         $driver->write_byte(0x10, $PCA9685{'AllLedOffH'});  # Orderly shutdown.
         undef($driver);
      }

      if ($main::MainRun > 2) {
         &DnB_Turnout::ProcessTurnoutFile($main::TurnoutFile, "Write",
                                          \%main::TurnoutData);
         sleep 1;   # Time for file system to complete.
      }
   }
   &DisplayMessage("$main::ChildName, ctrl+c initiated stop.");
   sleep 1;
   exit(0);
}

# ============================================================================
# FUNCTION:  ReadFile
#
# DESCRIPTION:
#    This routine reads the specified file into the specified array.
#
# CALLING SYNTAX:
#    $result = &ReadFile($InputFile, \@Array, "NoTrim");
#
# ARGUMENTS:
#    $InputFile      File to read.
#    \@Array         Pointer to array for output records.
#    $NoTrim         Suppress record trim following read.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ReadFile {

   my($InputFile, $OutputArrayPointer, $NoTrim) = @_;
   my($FileHandle, $ntry);

   &DisplayDebug(2, "ReadFile, Loading from $InputFile ...");

   unless (open($FileHandle, '<', $InputFile)) {
      &DisplayError("ReadFile, opening file for read: $InputFile - $!");
      return 1;
```

```
301      }
302      @$OutputArrayPointer = <$FileHandle>;
303      close($FileHandle);
304
305      unless ($NoTrim) {
306         foreach my $ntry (@$OutputArrayPointer) {
307            $ntry = Trim($ntry);
308         }
309      }
310      return 0;
311   }
312
313   # ============================================================================
314   # FUNCTION:  ReadBin
315   #
316   # DESCRIPTION:
317   #    This routine reads the specified binary file into the specified variable.
318   #
319   # CALLING SYNTAX:
320   #    $result = &ReadBin($Filename, \$BufferPntr);
321   #
322   # ARGUMENTS:
323   #    $Filename       File to read.
324   #    $BufferPntr     Pointer to variable.
325   #
326   # RETURNED VALUES:
327   #    0 = Success,  1 = Error.
328   #
329   # ACCESSED GLOBAL VARIABLES:
330   #    None.
331   # ============================================================================
332   sub ReadBin {
333      my($Filename, $BufferPntr) = @_;
334      my($FileHandle);
335
336      &DisplayDebug(2, "ReadBin, Filename: $Filename");
337
338      unless (open($FileHandle, '<', $Filename)) {
339         &DisplayError("ReadBin, opening file for read: $Filename - $!");
340         return 1;
341      }
342      binmode($FileHandle);
343      local $/ = undef;
344      $$BufferPntr = <$FileHandle>;
345      close($FileHandle);
346      &DisplayDebug(2, "ReadBin, length read: " . length($$BufferPntr));
347
348      return 0;
349   }
350
351   # ============================================================================
352   # FUNCTION:  ReadFileHandle
353   #
354   # DESCRIPTION:
355   #    This routine is used to perform sysread's of the specified number of
356   #    bytes from the specified file handle. The data is unpacked into a
357   #    character string; two characters per byte in hexidecimal format. The
358   #    returned length will always be the requested size times 2 plus the
359   #    length of the input $data contents. Any input $data from a previous
360   #    ReadBin call is prepended to the current data read.
```

```perl
#
# CALLING SYNTAX:
#    ($length, $data) = &ReadFileHandle($FileHandle, $size, $data);
#
# ARGUMENTS:
#    $FileHandle     Filehandle of input data.
#    $Size           Number of bytes to read from FileHandle.
#    $Data           Input $data contents, if any.
#
# RETURNED VALUES:
#    -1 = EOF,  length of data.
#    unpacked bytes read.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ReadFileHandle {

   my($FileHandle, $Size, $Data) = @_;
   my($sizeread, $newdata);

   &DisplayDebug(2, "ReadFileHandle, entry ...   Size: $Size");

   if ($Size > 0) {
      undef $/;
      $sizeread = sysread($FileHandle, $newdata, $Size);
      $/ = "\n";
      &DisplayDebug(2, "ReadFileHandle, sizeread: $sizeread");
      if ($sizeread > 0) {
         $newdata = unpack("H*", $newdata);
         $newdata = join("", $Data, $newdata);
         return (length($Data), $newdata);
      }
      else {
         return (-1, $Data);
      }
   }
   return (length($Data), $Data);
}

# ============================================================================
# FUNCTION:  WriteFile
#
# DESCRIPTION:
#    This routine writes the specified array to the specified file. If the file
#    already exists, it is deleted.
#
# CALLING SYNTAX:
#    $result = &WriteFile($OutputFile, \@Array, "Trim");
#
# ARGUMENTS:
#    $OutputFile     File to write.
#    $Array          Pointer to array for output records.
#    $Trim           Trim records before writing to file.
#
# RETURNED VALUES:
#    0 = Success,  exit code on Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
```

```perl
# ============================================================================
sub WriteFile {

   my($OutputFile, $OutputArrayPointer, $Trim) = @_;
   my($FileHandle);

   &DisplayDebug(2, "WriteFile, Creating $OutputFile ...");

   unlink ($OutputFile) if (-e $OutputFile);

   unless (open($FileHandle, '>', $OutputFile)) {
      &DisplayError("WriteFile, opening file for write: $OutputFile - $!");
      return 1;
   }
   foreach my $ntry (@$OutputArrayPointer) {
      $ntry = Trim($ntry) if ($Trim);
      unless (print $FileHandle $ntry, "\n") {
         &DisplayError("WriteFile, writing file: $OutputFile - $!");
         close($FileHandle);
         return 1;
      }
   }
   close($FileHandle);
   return 0;
}

# ============================================================================
# FUNCTION:  WriteFileAppend
#
# DESCRIPTION:
#    This routine writes the specified array to the specified file. If the file
#    already exists, the new data is appended to the current data.
#
# CALLING SYNTAX:
#    $result = &WriteFileAppend($OutputFile, \@Array, "Trim");
#
# ARGUMENTS:
#    $OutputFile     File to write.
#    $Array          Pointer to array for output records.
#    $Trim           Trim records before writing to file.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub WriteFileAppend {

   my($OutputFile, $OutputArrayPointer, $Trim) = @_;
   my($FileHandle);

   if (-e $OutputFile) {
      &DisplayDebug(2, "WriteFileAppend, Updating $OutputFile ...");
      unless (open($FileHandle, '>>', $OutputFile)) {
         &DisplayError("WriteFileAppend, opening file for append: " .
                       "$OutputFile - $!");
         return 1;
      }
   }
```

```
481      else {
482         &DisplayDebug(2, "WriteFileAppend: Creating $OutputFile ...");
483         unless (open($FileHandle, '>', $OutputFile)) {
484            &DisplayError("WriteFileAppend, opening file for write: $OutputFile - $!");
485            return 1;
486         }
487      }
488      foreach my $ntry (@$OutputArrayPointer) {
489         $ntry = Trim($ntry) if ($Trim);
490         unless (print $FileHandle $ntry, "\n") {
491            &DisplayError("WriteFileAppend, writing file: $OutputFile - $!");
492            close($FileHandle);
493            return 1;
494         }
495      }
496      close($FileHandle);
497      return 0;
498   }
499
500   # ============================================================================
501   # FUNCTION:  DisplayMessage
502   #
503   # DESCRIPTION:
504   #    Displays a message to the user. If variable $main::SerialPort is set,
505   #    the message is directed to the Raspberry Pi serial port.
506   #
507   # CALLING SYNTAX:
508   #    $result = &DisplayMessage($Message);
509   #
510   # ARGUMENTS:
511   #    $Message         Message to be output.
512   #
513   # RETURNED VALUES:
514   #    0 = Success,  1 = Error.
515   #
516   # ACCESSED GLOBAL VARIABLES:
517   #    $main::SerialPort, $main::Opt{q}
518   # ============================================================================
519   sub DisplayMessage {
520
521      my($Message) = @_;
522      my($time) = &DateTime('', '', '-');
523
524      if ($main::SerialPort > 0) {
525         $main::SerialPort->puts("$$ $time $Message\n");
526      }
527      else {
528         print STDOUT "$$ $time $Message\n" unless (defined($main::Opt{q}));
529      }
530      return 0;
531   }
532
533   # ============================================================================
534   # FUNCTION:  DisplayError
535   #
536   # DESCRIPTION:
537   #    Displays an error message to the user. If variable $main::SerialPort
538   #    is set, the message is directed to the Raspberry Pi serial port.
539   #
540   # CALLING SYNTAX:
```

```
#    $result = &DisplayError($Message, $Stdout);
#
# ARGUMENTS:
#    $Message         Message to be output.
#    $Stdout          Sends message to STDOUT if set.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::SerialPort, $main::Opt{q}
# ============================================================================
sub DisplayError {

   my($Message, $Stdout) = @_;
   my($time) = &DateTime('', '', '-');
   my($result);

   if ($main::SerialPort > 0) {
      $main::SerialPort->puts("$$ $time *** error: $Message\n");
   }
   else {
      unless (defined($main::Opt{q})) {
         if ($Stdout) {
            return (print STDOUT "$$ $time *** error: $Message\n");
         }
         else {
            return (print STDERR "$$ $time *** error: $Message\n");
         }
      }
   }
   return 0;
}

# ============================================================================
# FUNCTION:  DisplayWarning
#
# DESCRIPTION:
#    Displays a warning message to the user. If variable $main::SerialPort
#    is set, the message is directed to the Raspberry Pi serial port.
#
# CALLING SYNTAX:
#    $result = &DisplayWarning($Message, $Stdout);
#
# ARGUMENTS:
#    $Message       Message to be output.
#    $Stdout        Sends message to STDOUT if set.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::SerialPort, $main::Opt{q}
# ============================================================================
sub DisplayWarning {

   my($Message, $Stdout) = @_;
   my($time) = &DateTime('', '', '-');

   if ($main::SerialPort > 0 and $main::WiringApiObj ne "") {
```

```
601         $main::SerialPort->puts("$$ $time --> error: $Message\n");
602      }
603      else {
604         unless (defined($main::Opt{q})) {
605            if ($Stdout) {
606               return (print STDOUT "$$ $time --> warning: $Message\n");
607            }
608            else {
609               return (print STDERR "$$ $time --> warning: $Message\n");
610            }
611         }
612      }
613      return 0;
614   }
615
616   # ============================================================================
617   # FUNCTION:  DisplayDebug
618   #
619   # DESCRIPTION:
620   #    Displays a debug message to the user if the current program $DebugLevel
621   #    is >= to the message debug level. If variable $main::SerialPort is set,
622   #    the message is directed to the Raspberry Pi serial port.
623   #
624   # CALLING SYNTAX:
625   #    $result = &DisplayDebug($Level, $Message);
626   #
627   # ARGUMENTS:
628   #    $Level               Message debug level.
629   #    $Message             Message to be output.
630   #
631   # RETURNED VALUES:
632   #    0 = Success,  1 = Error.
633   #
634   # ACCESSED GLOBAL VARIABLES:
635   #    $main::SerialPort, $main::DebugLevel, $main::Opt{q}
636   # ============================================================================
637   sub DisplayDebug {
638
639      my($Level, $Message) = @_;
640      my($time) = &DateTime('', '', '-');
641
642      if ($main::DebugLevel >= $Level) {
643         if ($main::SerialPort > 0) {
644            $main::SerialPort->puts("$$ $time debug${Level}: $Message\n");
645         }
646         else {
647            unless (defined($main::Opt{q})) {
648               print STDOUT "$$ $time debug${Level}: $Message\n";
649            }
650         }
651      }
652      return 0;
653   }
654
655   # ============================================================================
656   # FUNCTION:  Trim
657   #
658   # DESCRIPTION:
659   #    Removes newline, leading, and trailing spaces from specified input. Input
660   #    string is returned.
```

```
#
# CALLING SYNTAX:
#    $String = &Trim($String);
#
# ARGUMENTS:
#    $String        String to trim.
#
# RETURNED VALUES:
#    Trimmed and chomped string.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ==========================================================================
sub Trim {

   my($String) = @_;

   chomp($String);                          # Remove trailing newline.
   $String =~ s/^\s+//;                     # Remove leading whitespace.
   $String =~ s/\s+$//;                     # Remove trailing whitespace.
   return($String);
}

# ==========================================================================
# FUNCTION:  TrimArray
#
# DESCRIPTION:
#    Removes leading and trailing blank lines from the specified array. The
#    array is specified by reference.
#
# CALLING SYNTAX:
#    $result = &TrimArray(\@array);
#
# ARGUMENTS:
#    \@array        Pointer reference to the array to be processed.
#
# RETURNED VALUES:
#    0 = Success,  1 = Array is empty.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ==========================================================================
sub TrimArray {

   my($arrayRef) = @_;

   splice(@$arrayRef, 0, 1) while ($#$arrayRef > 0 and $$arrayRef[0] =~ m/^\s*$/);
   splice(@$arrayRef, $#$arrayRef, 1) while ($#$arrayRef > 0 and
                                             $$arrayRef[$#$arrayRef] =~ m/^\s*$/);
   return 0;
}

# ==========================================================================
# FUNCTION: SplitIt
#
# DESCRIPTION:
#    This function is called to split the supplied string into parts using the
#    specified character as the separator character. The results are trimmed
#    of leading and trailing whitespace and returned in an array.
#
```

```perl
# CALLING SYNTAX:
#    @Array = &SplitIt($Char, $Rec);
#
# ARGUMENTS:
#    $Char        The separator character.
#    $Rec         The one-line record to be split.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub SplitIt {

   my($Char, $Rec) = @_;
   my(@temp, $i);

   if (($Char) and ($Rec)) {
      $Rec = &Trim($Rec);
      @temp = split($Char,$Rec);
      for ($i = 0; $i <= $#temp; $i++) {
         @temp[$i] = &Trim(@temp[$i]);
      }
      return @temp;
   }
   else {
      return $Rec;
   }
}

# ============================================================================
# FUNCTION:  HexToAscii
#
# DESCRIPTION:
#    This routine is used to convert a hex data string to its equivalent
#    ASCII characters. Two characters from the input data stream are used
#    for each output character.
#
# CALLING SYNTAX:
#    $AsciiStr = &HexToAscii($HexData);
#
# ARGUMENTS:
#    $HexData            Input hex data to convert.
#
# RETURNED VALUES:
#    ASCII character string
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub HexToAscii {

   my($HexData) = @_;
   my($x, $chr);  my($AsciiStr) = "";

   for ($x = 0; $x < length($HexData); $x += 2) {
      $chr = chr(hex(substr($HexData, $x, 2)));
      $AsciiStr = join("", $AsciiStr, $chr);
   }
   return $AsciiStr;
}

```

```perl
# ==========================================================================
# FUNCTION: DateTime
#
# DESCRIPTION:
#    This function, when called, returns a formatted date/time string for the
#    specified $Time. The current server time is used if not specified. The
#    arguments are used to affect how the date and time components are joined
#    into the result string. For example:
#
#    For $DateJoin = "-", $TimeJoin = ":", and $DatetimeJoin = "_", the returned
#    string would be: '2007-06-13_08:15:41'
#
# CALLING SYNTAX:
#    $datetime = DateTime($DateJoin, $TimeJoin, $DatetimeJoin, $Time);
#
# ARGUMENTS:
#    $DateJoin        Character string to join date components
#    $TimeJoin        Character string to join time components
#    $DatetimeJoin    Character string to join date and time components
#    $Time            Optional time to be converted
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ==========================================================================
sub DateTime {
   my($DateJoin, $TimeJoin, $DatetimeJoin, $Time) = @_;
   my($date, $time, $sec, $min, $hour, $day, $month, $year);

   if ($Time eq "") {
      ($sec, $min, $hour, $day, $month, $year) = localtime;
   }
   else {
      ($sec, $min, $hour, $day, $month, $year) = localtime($Time);
   }

   $month = $month+1;
   $month = "0".$month if (length($month) == 1);
   $day = "0".$day if (length($day) == 1);
   $year = $year + 1900;
   $hour = "0".$hour if (length($hour) == 1);
   $min = "0".$min if (length($min) == 1);
   $sec = "0".$sec if (length($sec) == 1);

   $date = join($DateJoin, $year, $month, $day);
   $time = join($TimeJoin, $hour, $min, $sec);
   return join($DatetimeJoin, $date, $time);
}

# ==========================================================================
# FUNCTION: DelDirTree
#
# DESCRIPTION:
#    This function recursively deletes directories and files in the specified
#    directory. The specified directory is then deleted.
#
# CALLING SYNTAX:
#    $result = DelDirTree($Dir);
#
# ARGUMENTS:
#    $Dir              Directory tree to be deleted.
```

```perl
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub DelDirTree {
   my($Dir) = @_;
   my($file);      my(@list) = ();

   &DisplayDebug(2, "DelDirTree, Entry ...   Dir: $Dir");

   unless (opendir(DIR, $Dir)) {
      &DisplayError("DelDirTree, opening directory: $Dir - $!");
      return 1;
   }
   @list = readdir(DIR);
   closedir(DIR);

   foreach my $ntry (@list) {
      next if (($ntry eq ".") or ($ntry eq ".."));  # Skip . and .. directories
      $file = join("\\", $Dir, $ntry);
      if (-d $file) {
         return 1 if (&DelDirTree($file));          # Recursion into directory
      }
      else {
         unless (unlink $file) {
            &DisplayError("DelDirTree, removing file: $file - $!");
            return 1;
         }
      }
   }
   unless (rmdir $Dir) {
      &DisplayError("DelDirTree, can't remove directory: $Dir - $!");
      return 1;
   }
   return 0;
}

# ============================================================================
# FUNCTION:  GrepFile
#
# DESCRIPTION:
#   Grep the specified file for the specified strings. This routine used instead
#   of a backtick/system command for platform portability.
#
#   The $Option specifies how the search string is used.
#      'single' - The string is used as specified. Default if not specified.
#      'multi'  - String is a space separated list of words. Any word matches.
#
# CALLING SYNTAX:
#   $result = &GrepFile($String, $File, $Option);
#
# ARGUMENTS:
#   $String        The string to search for.
#   $File          The file to search.
#   $Option        Search option.
#
# RETURNED VALUES:
```

```
#   Success: Matched line or "" if no match.
#
# ACCESSED GLOBAL VARIABLES:
#   None.
# ============================================================================
sub GrepFile {
   my($String, $File, $Option) = @_;
   my($FileHandle);
   my($grepResult, $prevLine) = ("","");

   &DisplayDebug(2,"GrepFile, String: '$String'  File: '$File'  Option: '$Option'");

   if (-e $File) {
      if (open($FileHandle, '<', $File)) {
         if ($Option =~ m/^m/) {
            $String =~ s#\s+#|#g;
            &DisplayDebug(2,"GrepFile, String: '$String'");
         }
         while (<$FileHandle>) {
            if ($_ =~ m/$String/) {
               $grepResult = $_;
               &DisplayDebug(2,"GrepFile, Matched: '$String'   " .
                               "grepResult: $grepResult");
               last;
            }
         }
         close($FileHandle);
      }
   }
   else {
      &DisplayError("GrepFile, file to grep not found: $File");
   }
   return Trim($grepResult);
}

# ============================================================================
# FUNCTION:  ShuffleArray
#
# DESCRIPTION:
#    This routine shuffles the specified array using the Fisher-Yates shuffle
#    algorithm. In plain terms, the algorithm randomly shuffles the sequence.
#
# CALLING SYNTAX:
#    $result = &ShuffleArray(\@Array);
#
# ARGUMENTS:
#    \@Array         Pointer to array to be shuffled.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ShuffleArray {
   my($Array) = @_;

   if ($#$Array > -1) {
      &DisplayDebug(3, "ShuffleArray, pre-shuffle : @$Array");
      my $i = @$Array;
```

```
961         while (--$i) {
962            my $j = int rand ($i + 1);
963            @$Array[$i,$j] = @$Array[$j,$i];
964         }
965         &DisplayDebug(3, "ShuffleArray, post-shuffle : @$Array");
966      }
967      return 0;
968   }
969
970   return 1;
971
```