```perl
# ============================================================================
# FILE: DnB_Message.pm                                              8/05/2020
#
# 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::ChildName, $main::$Opt{q}, %main::ServoBoardAddress

# ============================================================================
sub Ctrl_C {
```

```
241      my($driver, $I2C_Address);
242      my(%PCA9685) = ('ModeReg1' => 0x00, 'ModeReg2' => 0x01, 'AllLedOffH' => 0xFD,
243                      'PreScale' => 0xFE);
244
245      undef ($main::Opt{q});                        # Ensure console messages are on.
246      if ($main::ChildName eq 'Main') {
247         foreach my $key (sort keys(%main::ServoBoardAddress)) {
248            $I2C_Address = $main::ServoBoardAddress{$key};
249            $driver = RPi::I2C->new($I2C_Address);
250            unless ($driver->check_device($I2C_Address)) {
251               &DisplayError("Ctrl_C, Failed to instantiate I2C address: " .
252                             sprintf("0x%.2x",$I2C_Address));
253               next;
254            }
255            $driver->write_byte(0x10, $PCA9685{'AllLedOffH'});  # Orderly shutdown.
256            undef($driver);
257         }
258         $main::MainRun = 0;      # Stop the main loop.
259         return;
260      }
261      &DisplayMessage("$main::ChildName, ctrl+c initiated stop.");
262      exit(0);
263   }
264
265   # ============================================================================
266   # FUNCTION:  ReadFile
267   #
268   # DESCRIPTION:
269   #    This routine reads the specified file into the specified array.
270   #
271   # CALLING SYNTAX:
272   #    $result = &ReadFile($InputFile, \@Array, "NoTrim");
273   #
274   # ARGUMENTS:
275   #    $InputFile      File to read.
276   #    \@Array         Pointer to array for output records.
277   #    $NoTrim         Suppress record trim following read.
278   #
279   # RETURNED VALUES:
280   #    0 = Success,  1 = Error.
281   #
282   # ACCESSED GLOBAL VARIABLES:
283   #    None.
284   # ============================================================================
285   sub ReadFile {
286
287      my($InputFile, $OutputArrayPointer, $NoTrim) = @_;
288      my($FileHandle, $ntry);
289
290      &DisplayDebug(2, "ReadFile, Loading from $InputFile ...");
291
292      unless (open($FileHandle, '<', $InputFile)) {
293         &DisplayError("ReadFile, opening file for read: $InputFile - $!");
294         return 1;
295      }
296      @$OutputArrayPointer = <$FileHandle>;
297      close($FileHandle);
298
299      unless ($NoTrim) {
300         foreach my $ntry (@$OutputArrayPointer) {
```

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

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

```

```
421      &DisplayDebug(2, "WriteFile, Creating $OutputFile ...");
422
423      unlink ($OutputFile) if (-e $OutputFile);
424
425      unless (open($FileHandle, '>', $OutputFile)) {
426         &DisplayError("WriteFile, opening file for write: $OutputFile - $!");
427         return 1;
428      }
429      foreach my $ntry (@$OutputArrayPointer) {
430         $ntry = Trim($ntry) if ($Trim);
431         unless (print $FileHandle $ntry, "\n") {
432            &DisplayError("WriteFile, writing file: $OutputFile - $!");
433            close($FileHandle);
434            return 1;
435         }
436      }
437      close($FileHandle);
438      return 0;
439   }
440
441   # ============================================================================
442   # FUNCTION:  WriteFileAppend
443   #
444   # DESCRIPTION:
445   #    This routine writes the specified array to the specified file. If the file
446   #    already exists, the new data is appended to the current data.
447   #
448   # CALLING SYNTAX:
449   #    $result = &WriteFileAppend($OutputFile, \@Array, "Trim");
450   #
451   # ARGUMENTS:
452   #    $OutputFile     File to write.
453   #    $Array          Pointer to array for output records.
454   #    $Trim           Trim records before writing to file.
455   #
456   # RETURNED VALUES:
457   #    0 = Success,  1 = Error.
458   #
459   # ACCESSED GLOBAL VARIABLES:
460   #    None.
461   # ============================================================================
462   sub WriteFileAppend {
463
464      my($OutputFile, $OutputArrayPointer, $Trim) = @_;
465      my($FileHandle);
466
467      if (-e $OutputFile) {
468         &DisplayDebug(2, "WriteFileAppend, Updating $OutputFile ...");
469         unless (open($FileHandle, '>>', $OutputFile)) {
470            &DisplayError("WriteFileAppend, opening file for append: " .
471                          "$OutputFile - $!");
472            return 1;
473         }
474      }
475      else {
476         &DisplayDebug(2, "WriteFileAppend: Creating $OutputFile ...");
477         unless (open($FileHandle, '>', $OutputFile)) {
478            &DisplayError("WriteFileAppend, opening file for write: $OutputFile - $!");
479            return 1;
480         }
```

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

```
541  # RETURNED VALUES:
542  #    0 = Success,  1 = Error.
543  #
544  # ACCESSED GLOBAL VARIABLES:
545  #    $main::SerialPort, $main::Opt{q}
546  # ============================================================================
547  sub DisplayError {
548
549     my($Message, $Stdout) = @_;
550     my($time) = &DateTime('', '', '-');
551     my($result);
552
553     if ($main::SerialPort > 0) {
554        $main::SerialPort->puts("$$ $time *** error: $Message\n");
555     }
556     else {
557        unless (defined($main::Opt{q})) {
558           if ($Stdout) {
559              return (print STDOUT "$$ $time *** error: $Message\n");
560           }
561           else {
562              return (print STDERR "$$ $time *** error: $Message\n");
563           }
564        }
565        &PlaySound("A.wav",80);  # Sound error tone.
566     }
567     return 0;
568  }
569
570  # ============================================================================
571  # FUNCTION:  DisplayWarning
572  #
573  # DESCRIPTION:
574  #    Displays a warning message to the user. If variable $main::SerialPort
575  #    is set, the message is directed to the Raspberry Pi serial port.
576  #
577  # CALLING SYNTAX:
578  #    $result = &DisplayWarning($Message, $Stdout);
579  #
580  # ARGUMENTS:
581  #    $Message       Message to be output.
582  #    $Stdout        Sends message to STDOUT if set.
583  #
584  # RETURNED VALUES:
585  #    0 = Success,  1 = Error.
586  #
587  # ACCESSED GLOBAL VARIABLES:
588  #    $main::SerialPort, $main::Opt{q}
589  # ============================================================================
590  sub DisplayWarning {
591
592     my($Message, $Stdout) = @_;
593     my($time) = &DateTime('', '', '-');
594
595     if ($main::SerialPort > 0 and $main::WiringApiObj ne "") {
596        $main::SerialPort->puts("$$ $time --> error: $Message\n");
597     }
598     else {
599        unless (defined($main::Opt{q})) {
600           if ($Stdout) {
```

```
            return (print STDOUT "$$ $time --> warning: $Message\n");
         }
         else {
            return (print STDERR "$$ $time --> warning: $Message\n");
         }
      }
   }
   return 0;
}

# ============================================================================
# FUNCTION:  DisplayDebug
#
# DESCRIPTION:
#    Displays a debug message to the user if the current program $DebugLevel
#    is >= to the message debug level. If variable $main::SerialPort is set,
#    the message is directed to the Raspberry Pi serial port.
#
# CALLING SYNTAX:
#    $result = &DisplayDebug($Level, $Message);
#
# ARGUMENTS:
#    $Level               Message debug level.
#    $Message             Message to be output.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::SerialPort, $main::DebugLevel, $main::Opt{q}
# ============================================================================
sub DisplayDebug {

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

   if ($main::DebugLevel >= $Level) {
      if ($main::SerialPort > 0) {
         $main::SerialPort->puts("$$ $time debug${Level}: $Message\n");
      }
      else {
         unless (defined($main::Opt{q})) {
            print STDOUT "$$ $time debug${Level}: $Message\n";
         }
      }
   }
   return 0;
}

# ============================================================================
# FUNCTION:  Trim
#
# DESCRIPTION:
#    Removes newline, leading, and trailing spaces from specified input. Input
#    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.
#
# 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;
}

# ============================================================================
# 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.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
```

```perl
#    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.
# ============================================================================
```

```perl
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;
      while (--$i) {
         my $j = int rand ($i + 1);
         @$Array[$i,$j] = @$Array[$j,$i];
      }
      &DisplayDebug(3, "ShuffleArray, post-shuffle : @$Array");
```

```
961         }
962         return 0;
963     }
964
965     return 1;
966
```