```
#!/usr/local/bin/perl

use strict;
use warnings;

use Date::Manip;
use File::Copy;
use Mail::Sendmail;
use POSIX qw(strftime);
use Archive::Zip qw(:ERROR_CODES);

use CommonUtils;

#
# Configurable params
#

my $archive_dir  = "$base_dir/archive";
my $work_dir     = "$base_dir/working";
my $pending_dir  = "$base_dir/pending";
my $layout_dir   = "$base_dir/layouts";

#
# End of configurable params
#


my @now  = localtime();
my $YY   = strftime( "%y", @now );
my $YYYY = strftime( "%Y", @now );
my $MM   = strftime( "%m", @now );
my $MMM  = uc( strftime( "%b", @now ) );
my $DD   = strftime( "%d", @now );

my $start_time = strftime( "%Y-%m-%d  %H:%M:%S", @now );
my $end_time;

#
# archive_file
#-----------------------------------------------------
sub archive_file {
  my $src = shift;
  my $provider = shift;

  my $dir;

  $dir = "$archive_dir/$YYYY/$MM/$DD/$provider";

  bail( "Unable to create directory [$dir] ($!)" ) if system( "mkdir -p $dir" );

  copy($src, $dir) ||
    bail( "Unable to archive file [$src] ($!)" );

  return 1;
}

#
# count_recs
#-----------------------------------------------------
sub count_recs {

     my ($file, $layout) = @_;

     my $cnt;

     chomp($cnt = qx[fileinfo -l $layout -i '$file' -n 2>&1]);
```

```
     bail("\n\tUnable to get record count [$file]\n $cnt") if $cnt =~ /\D+/;

     return $cnt;
}



#
# create_file_info
#-------------------------------------------------------
sub create_file_info {
    my $dbh         = shift;
    my $zip_file    = shift;
    my $file        = shift;
    my $provider_id = shift;
    my $num_in      = shift;
    my $num_good    = shift;
    my $num_bad     = shift;
    my $num_dupe    = shift;

    my $sth;
    my $sql;
    my $file_info_id;


    # Create the file_info record for this file
    $sql = qq{ INSERT INTO file_info
               ( file_info_id, provider_id, zip_filename, filename,
                 gross_recs, good_recs, bad_recs, dupe_recs )
               VALUES( file_info_id_seq.nextval, ?, ?, ?, ?, ?, ?, ? ) };

    $dbh->do($sql, undef, $provider_id, "$zip_file", "$file", $num_in, $num_good, $
num_bad, $num_dupe) ||
        bail( "Unable to insert FileInfo [$sql] ($DBI::errstr)" );

    # Get the file_info_id that was just added
    $file_info_id = $dbh->selectrow_array( "SELECT file_info_id_seq.currval FROM du
al" ) ||
                     bail( "Unable to fetch the file_info_id ($DBI::errstr)" );


    # Create a RECEIVE action record
    $sql = qq{ INSERT INTO action
               ( action_id, file_info_id, action, action_dt, action_by )
               VALUES( action_id_seq.nextval, ?, 'RECEIVE', sysdate, 'system' ) };

    $dbh->do($sql, undef, $file_info_id) ||
        bail( "Unable to insert Action [$sql] ($DBI::errstr)" );

    $dbh->commit();

    return $file_info_id;
}



#
# pre_process_file
#
# unzip, cut out phone, dedupe, sort, and
# look for invalid phones
#
#-------------------------------------------------------
sub pre_process_file {
```

```
  my $dbh = shift;
  my $file = shift;
  my $provider_id = shift;
  my $prv = shift;

  my $sql;
  my $sth;
  my $cmd;

  my %mail;

  my $provider_name;
  my $provider;
  my $internal_file;
  my $file_type;
  my $raw_layout;
  my $incoming_layout;
  my $valid_layout;

  my $zip;
  my @zip_members;
  my $member;
  my $data_file;
  my $data_out_path;

  my $zip_file;   # input zip filename path information
  my $bad_file;   # bad records returned from sort
  my $dup_file;   # contains any duplicate records from exclude
  my $unq_file;   # contains unique records from exclude
  my $inv_file;   # contains records with invalid phone numbers
  my $good_file;  # contains records that are ready to be loaded
  my $in_file;    # basename of the input file

  my $num_in   = 0;
  my $num_bad  = 0;
  my $num_dupe = 0;
  my $num_good = 0;

  my $file_info_id;

  my @flds = qw(provider_name provider internal_file file_type raw_layout incoming_
layout valid_layout);

  ($provider_name, $provider, $internal_file,
   $file_type, $raw_layout,
   $incoming_layout, $valid_layout) = @{$prv->{$provider_id}}{@flds};

  # If file is zipped, then unzip it
  if ($file =~ /\.zip$/i)
  {
      my $found;

      $zip = Archive::Zip->new($file);
      bail("Unable to open .zip file [$file]") unless $zip;

      @zip_members = $zip->memberNames();

      # If the expected internal file is set, then check for it.
      if ($internal_file) {
          $found = ($data_file) = grep { /$internal_file/i } @zip_members;
      } else {
          if (@zip_members > 1) {
              bail("Too many files in .zip [$file]. Not sure which data file to use
." );
          }
          $found = 1;
```

```
            $data_file = $zip_members[0];
        }

        bail("Unable to find data file in .zip [$file]") unless $found;

        # unzip file
        $data_out_path = "$work_dir/" . lc( $data_file );
        if ($zip->extractMemberWithoutPaths( $data_file, $data_out_path ) != AZ_OK) {
            bail( "Unable to extract data file ($data_file) from ($file)." );
        }

        ($zip_file = $file) =~ s/^.*\/(.*?)$/$1/;

            unlink($file) ||
                bail("Unable to remove [$file] ($!)");
    } else {
        $data_out_path = $file;
        $zip_file = "";
    }

    if ($prv->{$provider_id}{cnv_process}) {
         my $cnvfile = "$data_out_path\.cnv";
         my $cmd = qq[$prv->{$provider_id}{cnv_process} $data_out_path $cnvfile];
         system($cmd) && bail ("\nUnable to run Cnv_Process\n[$cmd]\n");
         $data_out_path = $cnvfile;
    }

    # count the incoming records
    $num_in = count_recs($data_out_path, "$layout_dir/$raw_layout");

    # Convert from raw provider format to the incoming format and sort the phone file
.
    # Dedupe the sorted file.

    $bad_file = $data_out_path . ".bad";
    $dup_file = $data_out_path . ".dup";
    $unq_file = $data_out_path . ".unq";

    $incoming_layout = "$layout_dir/$incoming_layout";

    $cmd  = qq[sort -l $layout_dir/$raw_layout -i $data_out_path -L $incoming_layout
-k phone ];
    $cmd .= qq[-e $bad_file -o stdout | ];
    $cmd .= qq[exclude -l $incoming_layout -i stdin -L $incoming_layout ];
    $cmd .= qq[-u $dup_file -o $unq_file -k phone];

    system($cmd) && bail("Error sorting [$data_out_path]\n\n$cmd");

    archive_file($bad_file, $provider);
    archive_file($dup_file, $provider);

    # Check for invalid phone numbers
    $inv_file  = $data_out_path . ".inv";
    $good_file = $data_out_path . ".good";

    $cmd  = "cnv -l $layout_dir/$valid_layout -i $unq_file -L $incoming_layout -o $go
od_file";
    $cmd .= " -u $inv_file";

    system($cmd) && bail( "Error converting to remove invalid phones [$unq_file]" );


    # count the records for each step
    $num_bad  = count_recs($bad_file, $incoming_layout) + count_recs($inv_file, $inco
ming_layout);
    $num_dupe = count_recs($dup_file, $incoming_layout);
```

```
  $num_good = count_recs($good_file, $incoming_layout);

  # create a file_info record
  ( $in_file = $data_out_path ) =~ s/^.*\/(.*?)$/$1/;
  $file_info_id = create_file_info( $dbh, $zip_file, $in_file, $provider_id,
                                    $num_in, $num_good, $num_bad, $num_dupe );


  # stage file for loading
  copy( $good_file, "$pending_dir/$file_info_id.out" ) ||
    bail( "Unable to promote file [$good_file] to [$pending_dir/$file_info_id.out]
($!)" );


  # Cleanup
  unlink "$data_out_path" || bail( "Unable to delete data file [$data_out_path] ($!
)" );
  unlink "$bad_file"      || bail( "Unable to delete bad file [$bad_file] ($!)" );
  unlink "$inv_file"      || bail( "Unable to delete inv file [$inv_file] ($!)" );
  unlink "$dup_file"      || bail( "Unable to delete dup file [$dup_file] ($!)" );
  unlink "$unq_file"      || bail( "Unable to delete unq file [$unq_file] ($!)" );
  unlink "$good_file"     || bail( "Unable to delete good file [$good_file] ($!)" )
;


  # send 'file received' email and write out to report file
  @now = localtime();
  $end_time = strftime( "%Y-%m-%d  %H:%M:%S", @now );

  my $msg = qq{
  ACTION REQURED
  ==============
  This file must be approved before it will be loaded.
  Go to $url and click on Incoming.

  Start Time           : $start_time

  File Name            : $data_out_path
  File Type            : $file_type
  Provider Name        : $provider_name

  Gross Records        : $num_in
  Bad Records          : $num_bad
  Duplicate Records    : $num_dupe
  Net Records          : $num_good

  End Time             : $end_time
  };


  # Email the report
  $mail{From}    = $MailFrom;
  $mail{Subject} = "$provider_name Update file INCOMING [$file_info_id]";
  $mail{To}      = $email_list;
  $mail{Message} = "$msg\n\n";
  sendmail( %mail ) || die "Unable to email report [$Mail::Sendmail::error]";
}

#
# load_providers
#-------------------------------------------------------

sub load_providers {
    my $dbh = shift;

    my $prv  = {};
```

```
    my @flds = qw(
                    provider_id
                    provider_name
                    provider
                    internal_file
                    file_type
                    raw_layout
                    incoming_layout
                    valid_layout
                    incoming_dir
                    process
                );

    my $sql = "Select " . join(",", @flds) . " From provider";
    my $ar  = $dbh->selectall_arrayref($sql);
    shift @flds; #== Remove provider_id, used as key in $prv.
    map { @{ $prv->{$_->[0]} }{@flds} = @$_[1..$#{$_}] } @$ar;

    return $prv;
}

#
# MAIN
#-------------------------------------------------------

sub main
{
    my $sth;
    my $sql;
    my $provider_id;
    my $provider;
    my $incoming_dir;
    my $file;

    #== Note to MJD - Connect is through DBD::Proxy
    my $dbh = connect();
    my $prv = load_providers($dbh);

    bail("\n\tUnable to load Providers!\n") unless $prv;

    chdir $work_dir or bail("\n\tUnable to chdir() to $work_dir :\n$!");

    for my $id (keys %$prv) {
        $incoming_dir = $prv->{$id}{incoming_dir};
        unless (-e $incoming_dir) {
            bail( "\n\tUnable to create directory [$incoming_dir] : \n $!" ) if sys
tem("mkdir -p $incoming_dir");
            next;
        }

        opendir(INCOMING, $incoming_dir) ||
            bail("\n\tUnable to open directory [$incoming_dir] ($!)");

        while ( $file = readdir(INCOMING) ) {
            next if -d $file;
            next if ($file =~ /\.zip$/i) && system("unzip -v '$incoming_dir/$file'
2>&1 >/dev/null");

            copy("$incoming_dir/$file", $work_dir) ||
                bail("Unable to copy file [$file] to [$work_dir] ($!)");

            archive_file("$work_dir/$file", $prv->{$id}{provider});

            unlink( "$incoming_dir/$file" ) ||
                bail("Unable to remove incoming file [$file] ($!)");
```

```
            pre_process_file($dbh, "$work_dir/$file", $id, $prv);
        }

        close INCOMING;
    }
}

main();

exit;
```