1. EXEC PLOTTER

procedure TfrmMain.PLOT\_eqClick(Sender: TObject);

var

s : string;

i, j, iNumS : integer;

begin

frmMain.mmoResult.Lines.Clear;

s := frmMain.lblProgramPath.Caption + 'PLOTTER\PLOTTER1a.EXE';

ChDir(frmMain.lblProgramPath.Caption + 'PLOTTER');

// Utils.ExecProgram(s,'WAIT', SW\_Minimize);

UTILS.ExecProgram(s,'WAIT', SW\_Minimize);

frmMain.mmoResult.Lines.LoadFromFile(frmMain.lblProgramPath.Caption + 'PLOTTER\PLOT.OUT');

// ExtractResultClick(nil);

ChDir(frmMain.lblProgramPath.Caption);

end;

procedure ExecProgram(ProgramName: String; sWait: String; Visibility: Integer);

var

StartupInfo: TStartupInfo;

ProcInfo: TProcessInformation;

CreateOK: Boolean;

begin

// Ensure valid input for ProgramName

if ProgramName = '' then

begin

ShowMessage('Program name cannot be empty.');

Exit;

end;

// Initialize StartupInfo and ProcInfo to ensure clean state

ZeroMemory(@StartupInfo, SizeOf(TStartupInfo));

ZeroMemory(@ProcInfo, SizeOf(TProcessInformation));

StartupInfo.cb := SizeOf(TStartupInfo);

StartupInfo.dwFlags := STARTF\_USESHOWWINDOW;

StartupInfo.wShowWindow := Visibility;

// Attempt to create the process

CreateOK := CreateProcess(nil, PChar(ProgramName), nil, nil, False,

CREATE\_NEW\_PROCESS\_GROUP or NORMAL\_PRIORITY\_CLASS,

nil, nil, StartupInfo, ProcInfo);

if CreateOK then

begin

try

if sWait = 'WAIT' then

WaitForSingleObject(ProcInfo.hProcess, INFINITE);

finally

// Ensure that handles are closed properly

CloseHandle(ProcInfo.hProcess);

CloseHandle(ProcInfo.hThread);

end;

end

else

begin

ShowMessage('Unable to run ' + ProgramName);

end;

end;

1. EXTRACT RESULT HEADING

procedure ExtractResult\_hdg(EQP\_result : string);

var

i : integer;

s, x : string;

values: TArray<string>;

hr, min, sec : string;

LAT, LON, DEP : string;

year, mon, day : string;

SecondsFloat: Single;

Seconds, Milliseconds: Word;

begin

frmMain.ledRMS.Text := RightStr(EQP\_result,4);

s := EQP\_result;

values := s.Split([' '], TStringSplitOptions.ExcludeEmpty);

LAT := FormatFloat('00.00', StrToFloat(values[1])); // LAT

LON := FormatFloat('000.00', StrToFloat(values[2])); // LON

DEP := FormatFloat('000.0', StrToFloat(values[3])); // DEP

hr := FormatFloat('00', StrToFloat(values[4])); // hr

min := FormatFloat('00', StrToFloat(values[5])); // min

sec := FormatFloat('00.00', StrToFloat(values[6])); // sec

//----- extract date-time to format

//----- fs.ShortDateFormat := 'yyyy-MM-dd' + fs.LongTimeFormat := 'hh:mm:ss.zzzz';

year := RightStr(frmMain.mmoResult.Lines[0], 2); // year = YYYY

year := '20' + FormatFloat('00', StrToFloat(year));

mon := LeftStr(frmMain.mmoResult.Lines[0], 2);

mon := FormatFloat('00', StrToFloat(mon)); // year + month

s := S +'-'+ FormatFloat('00', StrToFloat(mon)); // year + month

day := copy(frmMain.mmoResult.Lines[0], 3, 2); // day

day := FormatFloat('00', StrToFloat(day)); // year + month + day

s := s +'-'+ FormatFloat('00', StrToFloat(day)); // year + month + day

s := year +'-'+ mon +'-'+ day +' '+ hr +':'+ min +':'+ sec ;

s := UTILS\_DateTime.ExtractDateTime\_heading(s); // datetime

frmMain.sgMainData.Cells[0,0] := s +' '+ LAT +' '+ LON +' '+ DEP;

// update values of Outputs: datetime, lat, Lon, Dep

frmMain.ledLat.Text := FormatFloat('000.000', StrToFloat(values[1])); // LAT

frmMain.ledLon.Text := FormatFloat('000.000', StrToFloat(values[2])); // LON

frmMain.ledDep.Text := FormatFloat('000.0', StrToFloat(values[3])); // DEP

frmMain.ledEQ\_name.Text := Year +'\_'+ Mon + Day +'\_'+ Hr + Min;

frmMain.dtpEQDateTime.DateTime := EncodeDateTime(StrToInt(Year), StrToInt(Mon), StrToInt(Day), StrToInt(Hr), StrToInt(Min), 0, 0);

end;

1. UPDATE RESIDUAL

procedure Update\_resPS\_delta;

var

i, j, row: Integer;

mmoLines: TStringList;

sta, resP, resS, sDelta: string;

found: Boolean;

procedure ExtractColumns(const input: string);

begin

sta := Trim(Copy(input, 1, 4));

resp := Trim(Copy(input, 23, 5));

ress := Trim(Copy(input, 29, 5));

sdelta :=Trim(Copy(input, 17, 5)); // Extract delta

end;

begin

mmoLines := TStringList.Create;

try

// Split the memo's text into lines

mmoLines.Text := frmMain.mmoResult.Text;

// Iterate through the lines of the memo's text, starting at line 9

for i := 8 to mmoLines.Count - 1 do

begin

// Extract columns based on fixed positions

ExtractColumns(mmoLines[i]);

// Search for the existing STA value in the stringgrid

found := False;

for j := 0 to frmMain.sgMainData.RowCount - 1 do

begin

if frmMain.sgMainData.Cells[0, j] = sta then

begin

found := True;

row := j;

break;

end;

end;

// If the STA value is found, update the RESp and RESs columns

if found then

begin

frmMain.sgMainData.Cells[2, row] := resP; // RESp

frmMain.sgMainData.Cells[5, row] := resS; // RESs

frmMain.sgMainData.Cells[8, row] := sDelta; // Delta

end;

end;

finally

mmoLines.Free;

end;

end;

1. COUNT USED PS

procedure CountUsed\_PS;

var

i, ctr, cntP, cntS : Integer;

totP, totS: Integer;

begin

cntP := 0;

cntS := 0;

totS := 0;

for i := 1 to frmMain.sgMainData.RowCount - 1 do

begin

if frmMain.sgMainData.Cells[0, i] = '' then break;

if frmMain.sgMainData.Cells[3, i] = '\*' then

Inc(cntP);

if frmMain.sgMainData.Cells[6, i] = '\*' then

Inc(cntS);

end;

totP := StrToInt(frmMain.edtTotal\_P.Text);

for i := 1 to frmMain.sgMainData.RowCount - 1 do

if frmMain.sgMainData.Cells[3, i] <> '' then

Inc(totS);

totS := StrToInt(frmMain.edtTotal\_S.Text);

frmMain.edtUsed\_P.Text := IntToStr(totP- cntP);

frmMain.edtUsed\_S.Text := IntToStr(totS- cntS);

end;