1. Extract PHASEDATA

procedure TfrmMain.PhaseData\_extractClick(Sender: TObject);

var

EQDate: TDateTime;

CSVFilePath, ProgramPath: string;

begin

EQDate := ISO8601ToDate(ledEQ\_datex.Text);

ProgramPath := lblProgramPath.Caption; // Path from lblProgramPath

// Call the conversion procedure

ConvertExcelToCSV(EQDate, ProgramPath);

CSVFilePath := IncludeTrailingPathDelimiter(ProgramPath) + 'TEMP\temp.csv';

// Load the CSV data into the TStringGrid (sgPhaseData)

LoadCSVToGrid(CSVFilePath, sgPhaseData);

end;

1. FILTER TIME (Phase Data):

procedure PhaseData\_Filter\_Time2(SourceGrid, TargetGrid: TStringGrid; InputDate, InputTime: string; CutoffMinutes: Integer);

var

RowIndex, TargetRowIndex: Integer;

PArrivalTimeStr: string;

ParsedDate, ParsedTime, InputDateTime, GridDateTime: TDateTime;

FormatSettings: TFormatSettings;

begin

// Set up custom format settings for date and time parsing

FormatSettings := TFormatSettings.Create;

FormatSettings.DateSeparator := '-';

FormatSettings.ShortDateFormat := 'yyyy-mm-dd';

FormatSettings.TimeSeparator := ':';

FormatSettings.ShortTimeFormat := 'hh:mm';

// Parse the input date (yyyy-mm-dd) into TDateTime

if not TryStrToDate(InputDate, ParsedDate, FormatSettings) then

raise Exception.Create('Invalid input date format');

// Parse the input time (hh:mm) into TDateTime (only the fractional part will be used)

if not TryStrToTime(InputTime, ParsedTime, FormatSettings) then

raise Exception.Create('Invalid input time format');

// Combine the parsed date and time into a single TDateTime

InputDateTime := ParsedDate + Frac(ParsedTime); // Frac extracts the time part

// Initialize TargetGrid (sgPhaseData\_Filter\_Time) and set the column count to match SourceGrid

TargetGrid.RowCount := 1; // Reset to 1 row to start adding filtered data

TargetGrid.ColCount := SourceGrid.ColCount;

// Copy headers from SourceGrid to TargetGrid (assuming headers are in row 0)

for RowIndex := 0 to SourceGrid.ColCount - 1 do

begin

TargetGrid.Cells[RowIndex, 0] := SourceGrid.Cells[RowIndex, 2];

end;

// Loop through each row in the SourceGrid

TargetRowIndex := 1; // Start target row index at 1 since row 0 is the header

for RowIndex := 3 to SourceGrid.RowCount - 1 do

begin

// Assuming the "P arrival" time is in column 4 of the grid

PArrivalTimeStr := SourceGrid.Cells[3, RowIndex]; // Extract "P arrival" time part

// Check if "P arrival" is empty

if Trim(PArrivalTimeStr) = '' then Break;

// Convert "P arrival" time string (HHMMSS.FF) to TDateTime

try

GridDateTime := EncodeTime(

StrToInt(Copy(PArrivalTimeStr, 1, 2)), // Hours

StrToInt(Copy(PArrivalTimeStr, 3, 2)), // Minutes

StrToInt(Copy(PArrivalTimeStr, 5, 2)), // Seconds

Round(StrToFloat('0.' + Copy(PArrivalTimeStr, 8, 2)) \* 1000) // Milliseconds

);

except

Continue; // Skip invalid times

end;

// Combine the parsed date with the P arrival time

GridDateTime := ParsedDate + Frac(GridDateTime);

// Check if the grid "P arrival" time is within the specified cutoff minutes after the input time

if (GridDateTime >= InputDateTime) and (GridDateTime <= IncMinute(InputDateTime, CutoffMinutes)) then

begin

// If the row matches the filter criteria, copy it to the TargetGrid

TargetGrid.RowCount := TargetRowIndex + 1; // Increase the row count in the target grid

TargetGrid.Rows[TargetRowIndex] := SourceGrid.Rows[RowIndex]; // Copy the entire row

Inc(TargetRowIndex); // Move to the next row in the target grid

end;

end;

e

1. FILTER DISTANCE (Phase Data)

procedure TfrmMain.PhaseData\_Filter\_DistanceClick(Sender: TObject);

var

eqMag: Single;

Cutoff\_Distance: Double;

begin

// Retrieve the earthquake magnitude from the labeled edit

eqMag := StrToFloat(ledEQ\_Mag.Text);

// Determine the cutoff distance based on the magnitude

if (eqMag >= 3.5) and (eqMag <= 3.9) then

Cutoff\_Distance := 300

else if (eqMag >= 3.0) and (eqMag <= 3.4) then

Cutoff\_Distance := 200

else if (eqMag >= 2.5) and (eqMag <= 2.9) then

Cutoff\_Distance := 100

else if (eqMag < 2.5) then

Cutoff\_Distance := 50;

// Now filter sgPhaseData2 by distance and save the result to sgPhaseData\_Filter\_Distance

UTILS\_PhaseData.PhaseData\_Filter\_Distance(sgPhaseData\_Filter\_Time, sgSOEPD\_stations, sgPhaseData\_Filter\_Distance,

ledEQ\_Lat, ledEQ\_Lon, ledEQ\_Dep, ledEQ\_Mag, Cutoff\_Distance);

1. CONVERT DATETIME:

procedure TfrmMain.ConvertDateTime\_formatClick(Sender: TObject);

var

Row: Integer;

TimeStr, FormattedDateTime: string;

begin

// Loop through all rows, starting from row 1 (assuming row 0 is the header)

for Row := 1 to sgPhaseData\_Filter\_Distance.RowCount - 1 do

begin

// Process column 3 (P arrival time)

TimeStr := sgPhaseData\_Filter\_Distance.Cells[3, Row];

if Trim(TimeStr) <> '' then

begin

try

FormattedDateTime := ConvertTimeWithDate(TimeStr, ledEQ\_datex);

// FormattedDateTime := ConvertTimeWithDate(TimeStr, ledEQ\_datex);

FormattedDateTime := Copy(FormattedDateTime, 1, Length(FormattedDateTime));

sgPhaseData\_Filter\_Distance.Cells[3, Row] := FormattedDateTime;

except

on E: Exception do

ShowMessage('Error on row ' + IntToStr(Row) + ' in column 3: ' + E.Message);

end;

end;

// Process column 5 (S arrival time)

TimeStr := sgPhaseData\_Filter\_Distance.Cells[5, Row];

if Trim(TimeStr) <> '' then

begin

try

FormattedDateTime := ConvertTimeWithDate(TimeStr, ledEQ\_datex);

// Drop the third decimal place if present

FormattedDateTime := Copy(FormattedDateTime, 1, Length(FormattedDateTime));

sgPhaseData\_Filter\_Distance.Cells[5, Row] := FormattedDateTime;

except

on E: Exception do

ShowMessage('Error on row ' + IntToStr(Row) + ' in column 5: ' + E.Message);

end;

end;

end;

// Optional: Display a message when the conversion is complete

// ShowMessage('DateTime conversion completed.');

end;

1. COMBINE DATA:

procedure TfrmMain.ConvertDateTime\_formatClick(Sender: TObject);

var

Row: Integer;

TimeStr, FormattedDateTime: string;

begin

// Loop through all rows, starting from row 1 (assuming row 0 is the header)

for Row := 1 to sgPhaseData\_Filter\_Distance.RowCount - 1 do

begin

// Process column 3 (P arrival time)

TimeStr := sgPhaseData\_Filter\_Distance.Cells[3, Row];

if Trim(TimeStr) <> '' then

begin

try

FormattedDateTime := ConvertTimeWithDate(TimeStr, ledEQ\_datex);

// FormattedDateTime := ConvertTimeWithDate(TimeStr, ledEQ\_datex);

FormattedDateTime := Copy(FormattedDateTime, 1, Length(FormattedDateTime));

sgPhaseData\_Filter\_Distance.Cells[3, Row] := FormattedDateTime;

except

on E: Exception do

ShowMessage('Error on row ' + IntToStr(Row) + ' in column 3: ' + E.Message);

end;

end;

// Process column 5 (S arrival time)

TimeStr := sgPhaseData\_Filter\_Distance.Cells[5, Row];

if Trim(TimeStr) <> '' then

begin

try

FormattedDateTime := ConvertTimeWithDate(TimeStr, ledEQ\_datex);

// Drop the third decimal place if present

FormattedDateTime := Copy(FormattedDateTime, 1, Length(FormattedDateTime));

sgPhaseData\_Filter\_Distance.Cells[5, Row] := FormattedDateTime;

except

on E: Exception do

ShowMessage('Error on row ' + IntToStr(Row) + ' in column 5: ' + E.Message);

end;

end;

end;

// Optional: Display a message when the conversion is complete

// ShowMessage('DateTime conversion completed.');

end;

1. UPDATE DATETIME PICKER

procedure UpdateDateTimePickerFromGrid;

var

DateTimeValue: TDateTime;

begin

try

// Extract the TDateTime from the grid cell (sgAtlasData\_Final.Cells[4,1])

DateTimeValue := ExtractDateTime(frmMain.sgAtlasData\_Final.Cells[4, 1]);

// Update the DateTimePicker with the extracted value

frmMain.dtpEQDateTime.DateTime := DateTimeValue;

// ShowMessage('DateTimePicker updated to: ' + DateTimeToStr(DateTimeValue));

except

on E: Exception do

begin

ShowMessage('Failed to update DateTimePicker: ' + E.Message);

end;

end;

end;