```
unit PWSReplicator;

interface

uses Classes, SysUtils, Math, LCRConfig, CostingSteps, LCRGlobals, VLSSystemData,
LCRCostVars,
     PWS_Sample_Bins, StrUtils, VLSSystemEntryPoints,
     DB, DBClient, VCL.FlexCel.Core, FlexCel.XLSAdapter, CSVSample, frmMain,
     System.Variants;
type

TPWSReplicator = class(TObject)
public
  SDWISFile: string;
  BaselineName: string;
  OptionName: string;
  SampleName: string;
  MakeProfileSample : boolean;
  SampRate : double;
  MinPerCategory: integer;
  NoReplication: boolean;
  Config: TLCRConfig;
  BaseCostSteps: TCostingSteps;
  LCRBaseCostVars: TCostVars;
  LCRBaseCostSteps: TCostingSteps;
  LCRRBaseCostVars: TCostVars;
  LCRRBaseCostSteps: TCostingSteps;
  ScenCostVars: TCostVars;
  ScenCostSteps: TCostingSteps;
  ProxyRecords: Boolean;
  SmallProxyPop: integer;
  PWS90PctBp1: integer;
  PWS90PctBp2: integer;
  UseSavedBins: boolean;
  LSLLevel: string;
  SmallProxyLabel: string;

  CategoryCount: array[1..2, 1..9, 1..2] of integer;
  CategoryRep: array[1..2, 1..9, 1..2] of integer;
  CategoryWeight: array[1..2, 1..9, 1..2] of double;

  CatConnectionSum: array[1..2, 1..9, 1..2] of double;
  CatConnectionCount: array[1..2, 1..9, 1..2] of integer;
  CatConnectionMean: array[1..2, 1..9, 1..2] of double;

  arrPBin: array[0..2, 0..2, 0..2] of double;
  arrPAdjBin: array[0..1, 1..1, 1..3] of double;
  arrPAdjBin5L: array[0..1, 1..1, 1..3] of double;

  BaselineSampleFilename, OptionSampleFilename: string;
```

```
  cdsPFAS_SDWIS: TClientDataset;
  cdsSavedLSLs: TClientDataset;
  cdsMissingSrcWater: TClientDataset;

  { SizeCatSort
    1   <=100
    2   101-500
    3   501-1,000
    4   1,001-3,300
    5   3,301-10,000
    6   10,001-50,000
    7   50,001-100,000
    8   100,001-1,000,000
    9   >1M (excluded) (included 3/19/18)

    GW or SW
    1   'Ground water'
    2   'Surface water'
  }

  procedure ReadSDWIS;
  procedure WriteLCRBaseline;
  procedure WriteLCRRBaseline;
  procedure WriteBaseline3;
  procedure WriteOption;

  constructor Create;
  destructor Destroy; override;
private
  VLSSystemData: TVLSSystemData;
  PWSSampleBins: TPWSSampleBins;
  VLSSystemEntryPoints: TVLSSystemEntryPoints;
  SLVlsLSLs: TStringList;

  function GetRepFactor(st, ss, sw: integer): integer;
  function GetWeight(st, ss, sw: integer): double;
  function GetEP(const ss,sw: integer): integer;
  function AssignBaselineBin(CCT, LSL: integer): integer; overload;
  function AssignOptionBin(CCT, LSL, baseBin: integer; Option: string): integer;
overload;
  function AssignBaselineBin(p_values: TPWS_p_valuesRec; LSL: integer; BaselineName:
string): integer; overload;
  function AssignOptionBin(LSL: integer; p_values: TPWS_p_valuesRec; baseBin:
integer; Option: string): integer; overload;
  function AssignOptionBin2(LSL: integer; p_values: TPWS_p_valuesRec; baseBin:
integer; Option: string): integer;

  procedure ReadPFASSDWIS;
  procedure MakeCdsPFAS;
```

```
  procedure ReadSavedLSLs;
  procedure MakeCdsSavedLSLs;
  procedure ReadMissingSrcWater;
  procedure MakeCdsMissingSrcWater;
  procedure MakeVLSEndPointsLSL(pwsid: string; p_values: TPWS_p_valuesRec);
end;

implementation


{ TPWSReplicator }

function TPWSReplicator.AssignBaselineBin(CCT, LSL: integer): integer;
var
  r: double;
  i, iBin: integer;
  tmp: double;
begin
  r := iiRandom(rrBase);

  iBin := 0;
  tmp:=0;
  i:=1;
  repeat
    tmp:=tmp+arrPBin[CCT, LSL, i-1];
    iBin:=i;
    inc(i);
    if i>length(arrPBin[CCT, LSL]) then break;
  until tmp>=r;

  Result := iBin;
end;

function TPWSReplicator.AssignBaselineBin(p_values: TPWS_p_valuesRec; LSL: integer;
BaselineName: string): integer;
var
  r: double;
  i, iBin: integer;
  tmp: double;
  arrPBin_loc: array[1..5] of double;
  baseBin: integer;
begin
  r := iiRandom(rrBase);

  arrPBin_loc[1] := p_values.ppBin1;
  arrPBin_loc[2] := p_values.ppBin2;
  arrPBin_loc[3] := p_values.ppBin3;
  arrPBin_loc[4] := p_values.ppBin4;
  arrPBin_loc[5] := p_values.ppBin5;
```

```
  iBin := 0;
  tmp:=0;
  i:=1;
  repeat
    tmp:=tmp+arrPBin_loc[i];
    iBin:=i;
    inc(i);
    if i>length(arrPBin_loc) then break;
  until tmp>=r;

  if BaselineName = 'LCRR' then
  begin
    baseBin := iBin;

    if LSL = 0 then
      iBin := baseBin
    else
    begin
      if baseBin = 1 then
        iBin := 1
      else
      if baseBin = 2 then begin
        if p_values.ppBin2 + p_values.ppBin1 <= p_values.ppAdjBin1 then iBin := 1
          else if r < ((p_values.ppAdjBin1 - p_values.ppBin1) / p_values.ppBin2)
then iBin := 1
        else iBin := 2;
      end
      else
      if baseBin = 3 then
      begin
        if p_values.ppBin3 + p_values.ppBin2 + p_values.ppBin1 <= p_values.ppAdjBin1
then iBin := 1
             else if r < ((p_values.ppAdjBin1 - p_values.ppBin1 - p_values.ppBin2)
/ p_values.ppBin3) then iBin := 1
           else if p_values.ppBin3 + p_values.ppBin2 <= p_values.ppAdjBin2 then
iBin := 2
           else if r < ((p_values.ppAdjBin2 - p_values.ppBin2) / p_values.ppBin3)
then iBin := 2
             else iBin := 3
      end
      else
      if baseBin = 4 then
      begin
           if p_values.ppBin4 + p_values.ppBin3 + p_values.ppBin2 + p_values.ppBin1
<= p_values.ppAdjBin1 then iBin := 1
             else if r < ((p_values.ppAdjBin1 - p_values.ppBin1 - p_values.ppBin2 -
p_values.ppBin3) / p_values.ppBin4) then iBin := 1
             else if p_values.ppBin4 + p_values.ppBin3 + p_values.ppBin2  <=
p_values.ppAdjBin2 then iBin := 2
           else if r < ((p_values.ppAdjBin2 - p_values.ppBin2 - p_values.ppBin3) /
```

```
p_values.ppBin4) then iBin := 2
              else if p_values.ppBin4 + p_values.ppBin3 <= p_values.ppAdjBin3 then
iBin := 3
              else if r < ((p_values.ppAdjBin3 - p_values.ppBin3) / p_values.ppBin4)
then iBin := 3
            else iBin := 4;
      end
      else
      if baseBin = 5 then
      begin
        if p_values.ppBin5 + p_values.ppBin4 + p_values.ppBin3 + p_values.ppBin2 +
p_values.ppBin1 <= p_values.ppAdjBin1 then iBin := 1
        else if r < ((p_values.ppAdjBin1 - p_values.ppBin1 - p_values.ppBin2 -
p_values.ppBin3 - p_values.ppBin4) / p_values.ppBin5) then iBin := 1
          else if p_values.ppBin5 + p_values.ppBin4 + p_values.ppBin3 +
p_values.ppBin2 <= p_values.ppAdjBin2 then iBin := 2
              else if r < ((p_values.ppAdjBin2 - p_values.ppBin2 - p_values.ppBin3 -
p_values.ppBin4) / p_values.ppBin5) then iBin := 2
            else if p_values.ppBin5 + p_values.ppBin4 + p_values.ppBin3 <=
p_values.ppAdjBin3 then iBin := 3
              else if r < ((p_values.ppAdjBin3 - p_values.ppBin3 - p_values.ppBin4)
/ p_values.ppBin5) then iBin := 3
              else if p_values.ppBin5 + p_values.ppBin4 <= p_values.ppAdjBin4 then
iBin := 4
              else if r < ((p_values.ppAdjBin4 - p_values.ppBin4) / p_values.ppBin5)
then iBin := 4
            else iBin := 5;
      end;
    end;
  end;

  Result := iBin;
end;

function TPWSReplicator.AssignOptionBin(CCT, LSL, baseBin: integer; Option: string):
integer;
var
  r,p: double;
  iBin: integer;
  pAdjBin1, pAdjBin2: double;
  pBin1, pBin2, pBin3: double;
begin
  r := iiRandom(rrScen);
  iBin := 0;

  if Option = 'OW' then
  begin
    if LSL = 0 then
      iBin := baseBin
    else begin
```

```
      if baseBin = 1 then
        iBin := 1
      else
      if baseBin = 2 then begin
        pAdjBin1 := arrPAdjBin[CCT, LSL, 1];
        pBin1 := arrPBin[CCT, LSL, 0];
        pBin2 := arrPBin[CCT, LSL, 1];
        p:=(pAdjBin1 - pBin1) / pBin2;
        if r<p then iBin:=1 else iBin:=2;
      end else begin
        pAdjBin2 := arrPAdjBin[CCT, LSL, 2];
        pBin2 := arrPBin[CCT, LSL, 1];
        pBin3 := arrPBin[CCT, LSL, 2];
        p:=(pAdjBin2 - pBin2) / pBin3;
        if r<p then iBin:=2 else iBin:=3;
      end;
    end;
  end
  else
  if Option = 'OW5L' then
  begin
    if LSL = 0 then
      iBin := baseBin
    else begin
      if baseBin = 1 then
        iBin := 1
      else
      if baseBin = 2 then begin
        pAdjBin1 := arrPAdjBin5L[CCT, LSL, 1];
        pBin1 := arrPBin[CCT, LSL, 0];
        pBin2 := arrPBin[CCT, LSL, 1];
        p:=(pAdjBin1 - pBin1) / pBin2;
        if r<p then iBin:=1 else iBin:=2;
      end else begin
        pAdjBin2 := arrPAdjBin5L[CCT, LSL, 2];
        pBin2 := arrPBin[CCT, LSL, 1];
        pBin3 := arrPBin[CCT, LSL, 2];
        p:=(pAdjBin2 - pBin2) / pBin3;
        if r<p then iBin:=2 else iBin:=3;
      end;
    end;
  end;

  Result := iBin;
end;

function TPWSReplicator.AssignOptionBin(LSL: integer; p_values: TPWS_p_valuesRec;
baseBin: integer; Option: string): integer;
var
  r,p: double;
```

```
  iBin: integer;
  p21, p31, p32, p41, p42, p43, p51, p52, p53, p54: double;
begin
  r := iiRandom(rrScen);
  iBin := 0;

  if LSL = 0 then
    iBin := baseBin
  else
  begin
    if baseBin = 1 then
      iBin := 1
    else
    if baseBin = 2 then begin
        if p_values.ppBin2 + p_values.ppBin1 <= p_values.ppAdjBin1 then iBin := 1
          else if r < ((p_values.ppAdjBin1 - p_values.ppBin1) / p_values.ppBin2)
then iBin := 1
        else iBin := 2;
    end
    else
    if baseBin = 3 then
    begin
        if p_values.ppBin3 + p_values.ppBin2 + p_values.ppBin1 <= p_values.ppAdjBin1
then iBin := 1
            else if r < ((p_values.ppAdjBin1 - p_values.ppBin1 - p_values.ppBin2) /
p_values.ppBin3) then iBin := 1
          else if p_values.ppBin3 + p_values.ppBin2 <= p_values.ppAdjBin2 then iBin
:= 2
            else if r < ((p_values.ppAdjBin2 - p_values.ppBin2) / p_values.ppBin3)
then iBin := 2
            else iBin := 3
    end
    else
    if baseBin = 4 then
    begin
            if p_values.ppBin4 + p_values.ppBin3 + p_values.ppBin2 + p_values.ppBin1
<= p_values.ppAdjBin1 then iBin := 1
            else if r < ((p_values.ppAdjBin1 - p_values.ppBin1 - p_values.ppBin2 -
p_values.ppBin3) / p_values.ppBin4) then iBin := 1
            else if p_values.ppBin4 + p_values.ppBin3 + p_values.ppBin2  <=
p_values.ppAdjBin2 then iBin := 2
            else if r < ((p_values.ppAdjBin2 - p_values.ppBin2 - p_values.ppBin3) /
p_values.ppBin4) then iBin := 2
            else if p_values.ppBin4 + p_values.ppBin3 <= p_values.ppAdjBin3 then
iBin := 3
            else if r < ((p_values.ppAdjBin3 - p_values.ppBin3) / p_values.ppBin4)
then iBin := 3
            else iBin := 4;
    end
    else
```

```
    if baseBin = 5 then
    begin
        if p_values.ppBin5 + p_values.ppBin4 + p_values.ppBin3 + p_values.ppBin2 +
p_values.ppBin1 <= p_values.ppAdjBin1 then iBin := 1
      else if r < ((p_values.ppAdjBin1 - p_values.ppBin1 - p_values.ppBin2 -
p_values.ppBin3 - p_values.ppBin4) / p_values.ppBin5) then iBin := 1
          else if p_values.ppBin5 + p_values.ppBin4 + p_values.ppBin3 +
p_values.ppBin2 <= p_values.ppAdjBin2 then iBin := 2
            else if r < ((p_values.ppAdjBin2 - p_values.ppBin2 - p_values.ppBin3 -
p_values.ppBin4) / p_values.ppBin5) then iBin := 2
            else if p_values.ppBin5 + p_values.ppBin4 + p_values.ppBin3 <=
p_values.ppAdjBin3 then iBin := 3
            else if r < ((p_values.ppAdjBin3 - p_values.ppBin3 - p_values.ppBin4) /
p_values.ppBin5) then iBin := 3
            else if p_values.ppBin5 + p_values.ppBin4 <= p_values.ppAdjBin4 then
iBin := 4
            else if r < ((p_values.ppAdjBin4 - p_values.ppBin4) / p_values.ppBin5)
then iBin := 4
            else iBin := 5;
    end;
  end;

  Result := iBin;
end;

function TPWSReplicator.AssignOptionBin2(LSL: integer; p_values: TPWS_p_valuesRec;
baseBin: integer; Option: string): integer;
var
  r,v,tot,r2: double;
  i: integer;
  PAdjBin, NewProbs, AdjBin, Bin: array[1..5] of double;
begin
  if LSL = 0 then begin
    result := baseBin;
    exit;
  end;

  r := iiRandom(rrScen);
  Result:=999;
  fillchar(PAdjBin, sizeOf(PAdjBin),0);
  AdjBin[1] := p_values.ppAdjBin1; AdjBin[2] := p_values.ppAdjBin2; AdjBin[3] :=
p_values.ppAdjBin3;
  AdjBin[4] := p_values.ppAdjBin4; AdjBin[5] := p_values.ppAdjBin5;
  Bin[1] := p_values.ppBin1; Bin[2] := p_values.ppBin2; Bin[3] := p_values.ppBin3;
  Bin[4] := p_values.ppBin4; Bin[5] := p_values.ppBin5;

  NewProbs[1] := 0;
  for i := 2 to 5 do
    NewProbs[i] := AdjBin[i] / (1- AdjBin[1]);
```

```
  //find movement to Bin 1 and adjust remaining mass---
  tot := 0;
  for i:=1 to BaseBin do begin
    tot := tot + Bin[i];
    if (tot<=AdjBin[1]) then begin
      //everyone moves so short circuit out of here...
      if (BaseBin = i) then begin
        result := 1;
        exit;
      end;
    end else begin
      // calc remainder and folks going to bin 1.
      v := Bin[i] - (tot - AdjBin[1]);
      if (v>0) then begin
        //test to see if the portion should move....
        if BaseBin<>i then continue;
        r2 := iiRandom(rrScen);
        if (r2 < v/Bin[i]) then begin
          result := 1;
          exit;
        end;
      end;
    end;
  end;

  //if we made it here - just distribute based on altered probs for 2-5
  v:=0;
  for i := 2 to 5 do v := v + NewProbs[i];
  PAdjBin[1] := 0;
  for i := 2 to 5 do PAdjBin[i] := PAdjBin[i-1] + NewProbs[i]/v;
  for i := 2 to BaseBin do begin
    if r<=PAdjBin[i] then begin
      result := i;
      exit;
    end;
    Result:=5; //?????  Fix this nonsense....
  end;
end;

constructor TPWSReplicator.Create;
begin
  fillchar(CategoryCount, SizeOf(CategoryCount), 0);
  fillchar(CategoryRep, SizeOf(CategoryRep), 0);
  fillchar(CategoryWeight, SizeOf(CategoryWeight), 0);

  fillchar(CatConnectionSum, SizeOf(CatConnectionSum), 0);
  fillchar(CatConnectionCount, SizeOf(CatConnectionCount), 0);
  fillchar(CatConnectionMean, SizeOf(CatConnectionMean), 0);

// The following values are also in CostingSteps.pas TCostingSteps.Create
```

```
// and in the Variables database
  // arrPBin[CCT, LSL, Probs]
  arrPBin[0,0,0] := 0.023;
  arrPBin[0,0,1] := 0.028;
  arrPBin[0,0,2] := 0.948;
  arrPBin[1,0,0] := 0.033;
  arrPBin[1,0,1] := 0.020;
  arrPBin[1,0,2] := 0.947;
  arrPBin[0,1,0] := 0.027;
  arrPBin[0,1,1] := 0.060;
  arrPBin[0,1,2] := 0.913;
  arrPBin[1,1,0] := 0.014;
  arrPBin[1,1,1] := 0.081;
  arrPBin[1,1,2] := 0.905;

  // arrPAdjBin[CCT, LSL, Probs]
  arrPAdjBin[0, 1, 1] := 0.07;
  arrPAdjBin[0, 1, 2] := 0.06;
  arrPAdjBin[0, 1, 3] := 0.87;
  arrPAdjBin[1, 1, 1] := 0.07;
  arrPAdjBin[1, 1, 2] := 0.15;
  arrPAdjBin[1, 1, 3] := 0.78;

  // arrPAdjBin5L[CCT, LSL, Probs]
  arrPAdjBin5L[0, 1, 1] := 0.13;
  arrPAdjBin5L[0, 1, 2] := 0.15;
  arrPAdjBin5L[0, 1, 3] := 0.72;
  arrPAdjBin5L[1, 1, 1] := 0.22;
  arrPAdjBin5L[1, 1, 2] := 0.19;
  arrPAdjBin5L[1, 1, 3] := 0.59;

  VLSSystemData := TVLSSystemData.Create;
  PWSSampleBins := TPWSSampleBins.Create;
  VLSSystemEntryPoints := TVLSSystemEntryPoints.Create;

  BaselineSampleFilename := '';
  OptionSampleFilename := '';

  cdsPFAS_SDWIS := TClientDataset.Create(nil);
  cdsMissingSrcWater := TClientDataset.Create(nil);

  SLVlsLSLs := TStringList.Create;
  SLVlsLSLs.Delimiter := ',';
  SLVlsLSLs.StrictDelimiter := true;
  SLVlsLSLs.Add('PWSID,RecNo,LSL_vls,num_connect_vls')
end;

destructor TPWSReplicator.Destroy;
begin
  VLSSystemData.Free;
```

```
  PWSSampleBins.Free;
  VLSSystemEntryPoints.Free;

  cdsPFAS_SDWIS.Close;
  cdsPFAS_SDWIS.Free;
  cdsMissingSrcWater.Close;
  cdsMissingSrcWater.Free;

  if config.UseSavedLSLs = 1 then
  begin
    cdsSavedLSLs.Close;
    cdsSavedLSLs.Free;
  end;

  SLVlsLSLs.Free;

  inherited;
end;

function TPWSReplicator.GetEP(const ss, sw: integer): integer;
var i : integer;
    r,c : double;
begin
  Result:=0;

  if ss = 8 then begin
    Result := -1;
    exit;
  end;

  r:=Random;
  c:=0;
  for i:=1 to 50 do begin
    c:=c+Config.EntryPointProbs[sw,ss,i];
    if c>r then begin
      Result:=i;
      break;
    end;
  end;
  if NoRandom then Result:=1;
end;

function TPWSReplicator.GetRepFactor(st, ss, sw: integer): integer;
begin
  if NoReplication then
    Result := 1
  else
    Result := CategoryRep[st, ss, sw];
end;
```

```
function TPWSReplicator.GetWeight(st, ss, sw: integer): double;
begin
  Result := CategoryWeight[st, ss, sw];
end;

procedure TPWSReplicator.MakeCdsSavedLSLs;
begin
  with cdsSavedLSLs do begin
    FieldDefs.Add('PWSID', ftString, 25);
    FieldDefs.Add('LSL', ftInteger);

    IndexDefs.Add('IdxAll','PWSID',[]);

    CreateDataSet;
    LogChanges := False;
  end;
end;

procedure TPWSReplicator.MakeVLSEndPointsLSL(pwsid: string; p_values:
TPWS_p_valuesRec);
var
  r: double;
  rr, i, iLSL: integer;
  tmp: double;
  arr_loc: array[1..3] of double;

  LSL_vls: integer;
  num_connect_vls: integer;
  retLSL: integer;
  sLine: string;
  RecNo: integer;
begin
  for rr := 2 to VLSSystemEntryPoints.Xls.RowCount do
  begin
    if pwsid <> VLSSystemEntryPoints.Xls.GetStringFromCell(rr,1) then continue;

    RecNo := VLSSystemEntryPoints.xls.GetCellValue(rr, 16).AsVariant;

    LSL_vls := VLSSystemEntryPoints.xls.GetCellValue(rr, 10).AsVariant;
    num_connect_vls := VLSSystemEntryPoints.xls.GetCellValue(rr, 19).AsVariant;

    sLine := pwsid + ',' + RecNo.ToString + ',' + LSL_vls.ToString + ',' +
num_connect_vls.ToString;
    SLVlsLSLs.Add(sLine);
  end;
end;

procedure TPWSReplicator.MakeCdsMissingSrcWater;
begin
  with cdsMissingSrcWater do begin
```

```
    FieldDefs.Add('PWSID', ftString, 25);
    FieldDefs.Add('PrimarySource', ftString, 5);

    IndexDefs.Add('IdxAll','PWSID',[]);

    CreateDataSet;
    LogChanges := False;
  end;
end;

procedure TPWSReplicator.MakeCdsPFAS;
begin
  with cdsPFAS_SDWIS do begin
    FieldDefs.Add('PWSID', ftString, 25);
    FieldDefs.Add('EntryPoints', ftInteger);

    IndexDefs.Add('IdxAll','PWSID',[]);

    CreateDataSet;
    LogChanges := False;
  end;
end;

procedure TPWSReplicator.ReadMissingSrcWater;
var
  r: integer;
  PWSId: string;
  Xls: TExcelFile;
begin
  if SDWISFile = '' then
    exit;

  Xls := TXlsFile.Create(SDWISFile, False);
  Xls.ActiveSheetByName := 'MissingSourceWater';

  MakeCdsMissingSrcWater;

  for r := 2 to Xls.RowCount do begin
    PWSId := XLS.GetStringFromCell(r,1);

    cdsMissingSrcWater.Append;

    cdsMissingSrcWater.FieldByName('PWSID').AsString := PWSId;
    cdsMissingSrcWater.FieldByName('PrimarySource').AsString :=
Xls.GetStringFromCell(r,4);
  end;
end;

procedure TPWSReplicator.ReadPFASSDWIS;
var
```

```
  r: integer;
  PWSId: string;
  PFASXls: TExcelFile;
begin
  if SDWISFile = '' then
    exit;

  PFASXls := TXlsFile.Create(SDWISFile, False);
  PFASXls.ActiveSheetByName := 'PFAS_SDWIS';

  MakeCdsPFAS;

  for r := 2 to PFASXls.RowCount do begin
    PWSId := PFASXLS.GetStringFromCell(r,2);
    if length(PWSId) < 9 then PWSId := '0' + PWSId;

    cdsPFAS_SDWIS.Append;

    cdsPFAS_SDWIS.FieldByName('PWSID').AsString := PWSId;
    cdsPFAS_SDWIS.FieldByName('EntryPoints').AsInteger :=
PFASxls.GetCellValue(r,14).AsVariant;
  end;
end;

procedure TPWSReplicator.ReadSavedLSLs;
var
  slData: TStringList;
  i: integer;
  CSVSample: TCSVSample;
  sValues: string;
  pwsid, hpwsid: string;
begin
  slData := TStringList.Create;
  SLData.Delimiter := ',';
  SLData.StrictDelimiter := true;

  CSVSample := TCSVSample.Create;
  CSVSample.Filename := Config.SavedLSLsFilename;
  CSVSample.Column := 'PWSId,LSL';
  CSVSample.readSample;

  MakeCdsSavedLSLs;

  hpwsid := '';
  for i := 0 to CSVSample.slValues.Count-1 do
  begin
    sValues := CSVSample.slValues.Strings[i];
    slData.CommaText := sValues;

    pwsid := RemoveProxyLetter(slData.Strings[0]);
```

```
    if pwsid <> hpwsid then
    begin
      cdsSavedLSLs.Append;

      cdsSavedLSLs.FieldByName('PWSID').AsString := pwsid;
      cdsSavedLSLs.FieldByName('LSL').AsInteger := slData.Strings[1].ToInteger;
    end;
    hpwsid := pwsid;
  end;

  CSVSample.Free;
end;

procedure TPWSReplicator.ReadSDWIS;
var
  Xls: TExcelFile;
  r: integer;
  i, j, st: integer;

  SizeCatSort: integer;
  GwSw: string;
  PWSType: string;
  SrcWater: integer;
  PWSId: string;
  SystemType: integer;

  num_pop: integer;
  connections: integer;
  num_pop_connection: double;
  tmpstr: string;
  tmpfloat: double;
  bFound: boolean;
begin
  // This read is mainly to get the replication factors and weights.
  // It also creates client datasets for the PFAS data to get the entry points and
  // for the missing source water types.
  if SDWISFile = '' then
    exit;

  if Config.UseSavedLSLs = 1 then
  begin
    cdsSavedLSLs := TClientDataset.Create(nil);
    ReadSavedLSLs;
  end;

  ReadPFASSDWIS;
  ReadMissingSrcWater;
  cdsMissingSrcWater.IndexName := 'IdxAll';

  Xls := TXlsFile.Create(SDWISFile, False);
```

```
  Xls.ActiveSheetByName := 'Inventory Characteristics';

  {
  PWS ID        1
  PWS Name      2
  PWS Type      3
  Primary Source        4
  Is Wholesaler 5
  Activity Status       6
  Population Served Count       7
  Size Category 8
  Is School Or Daycare  9
  Service Connections Count     10
  Activity Status Code  11
  GW or SW      12
  SizeCatSort   13
  Primary Service Area  14
  Primary Service Area Type Code        15
  First Reported Date   16
  EPA Region    17
  Primacy Agency        18
  Owner Type    19
  Admin Name    20
  Org Name      21
  Phone Number  22
  Address Line1 23
  Address Line2 24
  City Name     25
  Administrative State Code     26
  System Location       State 27
  Zip Code      28
  LSLR_Milestone        29
  DEEMWQP       30
  DONEWQP       31
  Vio58 SOX or EOX?     32
  Vio59 33
  MobileHomePark        34
  Tribal_Flag   35
  9YrLcrWaiver  36
  PurchasesCCT  37
  CCT_YN (Used for CCT Status Determination)    38
  TrueB3        39
  Size_SourceWater      40
  Listed in SDWIS/Fed as Using Inhibitor Orthophosphate?        41
}

  for r := 2 to Xls.RowCount do begin
     // exclude systems serving > 1,000,000
     SizeCatSort := xls.GetCellValue(r, 13).AsVariant;
```

```
PWSId := xls.GetStringFromCell(r, 1);
// Exclude PWS JOINT REGIONAL WATER SUPPLY SYSTEM CA
if PWSId = 'CA3010071' then
  continue;

// exclude invalid GW or SW Code
GwSw := xls.GetStringFromCell(r, 12);

if GwSw = 'Groundwater' then
  GwSw := 'GW' else
if GwSw = 'Surface water' then
  GwSw := 'SW';

if GwSw = '-' then
begin
  bFound := false;
  bFound := cdsMissingSrcWater.FindKey([PWSId]);
  if bFound then
    GwSw := cdsMissingSrcWater.FieldByName('PrimarySource').AsString
  else
    raise Exception.Create('Primary water source not found for: ' + PWSId)
end;

if GwSW = 'GW' then SrcWater := 1
else SrcWater := 2;

PWSType := xls.GetStringFromCell(r, 3);

if PWSType = 'Community water system' then
  PWSType := 'CWS' else
if PWSType = 'Non-Transient non-community system' then
  PWSType := 'NTNCWS';

if PWSType = 'CWS' then
  SystemType := 1
else
  SystemType := 2;

tmpstr := xls.GetCellValue(r, 7).AsVariant;
num_pop := tmpstr.Replace(',','', [rfReplaceAll, rfIgnoreCase]).ToInteger();
if num_pop < 25 then num_pop := 25;

tmpstr := xls.GetCellValue(r, 10).AsVariant;
tmpfloat := tmpstr.Replace(',','', [rfReplaceAll, rfIgnoreCase]).ToDouble();
connections := round(tmpfloat);

if connections > 0 then
  num_pop_connection := num_pop / connections
else
  num_pop_connection := 0;
```

```
     CategoryCount[SystemType, SizeCatSort, SrcWater] := CategoryCount[SystemType,
SizeCatSort, SrcWater] + 1;

     if (num_pop_connection >= 1) and (num_pop_connection <= 5) then
     begin
       CatConnectionSum[SystemType, SizeCatSort, SrcWater] :=
CatConnectionSum[SystemType, SizeCatSort, SrcWater] + num_pop_connection;
       CatConnectionCount[SystemType, SizeCatSort, SrcWater] :=
CatConnectionCount[SystemType, SizeCatSort, SrcWater] + 1;
     end;
  end;

  FreeAndNil(Xls);

  for st := 1 to 2 do
    for i := 1 to 9 do
      for j := 1 to 2 do
      begin
        if CategoryCount[st,i,j] > 0 then CategoryRep[st,i,j] :=
System.Math.Ceil(MinPerCategory / CategoryCount[st,i,j]);
        if NoReplication then
          CategoryWeight[st,i,j] := 1
        else
        begin
          if CategoryRep[st,i,j] > 0 then
            CategoryWeight[st,i,j] := 1 / CategoryRep[st,i,j]
          else
            CategoryWeight[st,i,j] := 1;
        end;

        if CatConnectionCount[st,i,j] > 0 then
          CatConnectionMean[st,i,j] := CatConnectionSum[st,i,j] /
CatConnectionCount[st,i,j]
        else
          CatConnectionMean[st,i,j] := 0;
      end;
end;

procedure TPWSReplicator.WriteLCRBaseline;
var
  XlsIn: TExcelFile;
  r: integer;

  SizeCatSort: integer;
  GwSw: string;
  PWSType: string;
  SrcWater: integer;
  PWSId: string;
  CCT: integer;
```

```
  population: integer;
  connections: integer;
  first_ale: integer;
  numeps: integer;
  SystemType: integer;

  p_lsl: integer;
  num_lsl: double;
  pbasepo4, pbaseph, pbasephpo4, baselinepo4dose: integer;
  baselineph_wph, baselineph_woph, baselineph_wocct, baselineph_wpo4ph: integer;

  num_pop_connection: double;
  correct_num_pop_connection: double;
  num_connect: integer;
  tmpstr: string;
  tmpfloat: double;

  Outfile: string;

  cv: TCostVar;
  vname: string;
  curval, rawval: double;
  rCol, j, k, iRep: integer;

  Writer: TStreamWriter;
  sLine, sLine1, sLine2, sLine3: string;

  RepRates: TDoubleArray;
  SrcRepRates: TDoubleArray;
  SigRepRates: TDoubleArray;
  TrtRepRates: TDoubleArray;
  LSLVolPctRates: TDoubleArray;

  Bin: integer;
  BinChg: integer;

  bBin: integer;
  LCR_Bin, LCR_P90: integer;

  SLSavedData: TStringList;
  bFound: boolean;
  tPWSId: string;

  procedure WriteLine(sLine1, sLine2, sLine3: string; Bin, SmallCorrect, NumProxies:
integer; suffix: string);
  var
    sLineOut: string;
  begin

    sLineOut := sLine1 + suffix + ',';
```

```
    sLineOut := sLineOut + sLine2;
    sLineOut := sLineOut + Bin.ToString + ',';
    sLineOut := sLineOut + SmallCorrect.ToString + ',';
    sLineOut := sLineOut + NumProxies.ToString + ',';
    sLineOut := sLineOut + sLine3;

    sLineOut := Copy(sLineOut, 1, Length(sLineOut)-1);
    Writer.WriteLine(sLineOut);
  end;
begin
  // This is for the LCR baseline
  // Read the SDWIS sample workbook

  if UseSavedBins then
  begin
    PWSSampleBins.ReadBinFile(true, LSLLevel, SmallProxyLabel);
  end;

  // original SDWIS sample file
  XlsIn := TXlsFile.Create(SDWISFile, False);
  XlsIn.ActiveSheetByName := 'Inventory Characteristics';

  OutFile := ExtractFilePath(SDWISFile) + 'LCRBaseline_' + SampleName + '.csv';
  BaselineSampleFilename := OutFile;
  Writer := TStreamWriter.Create(OutFile);

  SystemType := 0;
  SizeCatSort := 0;
  SrcWater := 0;
  connections := 0;
  numeps := 0;

  cdsPFAS_SDWIS.IndexName := 'IdxAll';
  if Config.UseSavedLSLs = 1 then
    cdsSavedLSLs.IndexName := 'IdxAll';
  cdsMissingSrcWater.IndexName := 'IdxAll';

  for r := 1 to XlsIn.RowCount do begin
     // skip certain rows from the SDWIS sample for different reasons
     if r > 1 then
     begin
       if MakeProfileSample then begin
         if Random>SampRate then continue;
       end;

       SizeCatSort := XlsIn.GetCellValue(r, 13).AsVariant;

       PWSId := XlsIn.GetStringFromCell(r, 1);

       // Exclude PWS JOINT REGIONAL WATER SUPPLY SYSTEM CA
```

```
        if PWSId = 'CA3010071' then
          continue;

        // exclude invalid GW or SW Code
        GwSw := xlsIn.GetStringFromCell(r, 12);

        if GwSw = 'Groundwater' then
          GwSw := 'GW' else
        if GwSw = 'Surface water' then
          GwSw := 'SW';

        if GwSw = '-' then
        begin
          bFound := false;
          bFound := cdsMissingSrcWater.FindKey([PWSId]);
          if bFound then
            GwSw := cdsMissingSrcWater.FieldByName('PrimarySource').AsString
          else
            raise Exception.Create('Primary water source not found for: ' + PWSId)
        end;

        if GwSw = 'GW' then SrcWater := 1
        else SrcWater := 2;

        PWSType := xlsIn.GetStringFromCell(r, 3);

        if PWSType = 'Community water system' then
          PWSType := 'CWS' else
        if PWSType = 'Non-Transient non-community system' then
          PWSType := 'NTNCWS';

        if PWSType = 'CWS' then
          SystemType := 1
        else
          SystemType := 2;

        if xlsIn.GetStringFromCell(r, 38) = 'FALSE' then
          CCT := 0
        else
          CCT := 1;

        first_ale := 0;

        tmpstr := xlsIn.GetCellValue(r, 7).AsVariant;
        population := tmpstr.Replace(',','', [rfReplaceAll,
rfIgnoreCase]).ToInteger();
        if population < 25 then population := 25;

        tmpstr := xlsIn.GetCellValue(r, 10).AsVariant;
        tmpfloat := tmpstr.Replace(',','', [rfReplaceAll, rfIgnoreCase]).ToDouble();
```

```
        connections := round(tmpfloat);

        if connections > 0 then
          num_pop_connection := population / connections
        else
          num_pop_connection := 0;

        if (num_pop_connection >= 1) and (num_pop_connection <= 5) then
          correct_num_pop_connection := num_pop_connection
        else
          correct_num_pop_connection := CatConnectionMean[SystemType, SizeCatSort,
SrcWater];

        if correct_num_pop_connection > 0 then
          num_connect := round(population / correct_num_pop_connection)
        else
          num_connect := 1;

        if SystemType = 2 then
        begin
          num_connect := connections;
          if num_connect >= 20 then
            num_connect := 20;
        end;
      end;  // end skip some rows...

      // generate column names for sample file
      if r = 1 then
      begin
        sLine := '';
        sLine := sLine + 'PWSID,';
        sLine := sLine + 'SystemType,';
        sLine := sLine + 'Population,';
        sLine := sLine + 'Connections,';
        sLine := sLine + 'SourceWater,';
        sLine := sLine + 'CCT,';
        sLine := sLine + 'PopCat,';
        sLine := sLine + 'State,';
        sLine := sLine + 'Owner,';
        sLine := sLine + 'First_ale,';
        sLine := sLine + 'Weight,';
        sLine := sLine + 'NumberEPs,';
        sLine := sLine + 'LSL,';
        sLine := sLine + 'NumberLSLs,';
        sLine := sLine + 'CCTPO4,';
        sLine := sLine + 'CCTPH,';
        sLine := sLine + 'CCTBoth,';
        sLine := sLine + 'BaselinePO4Dose,';
        sLine := sLine + 'Baselineph_wPh,';
        sLine := sLine + 'Baselineph_woPh,';
```

```
sLine := sLine + 'Baselineph_woCCT,';
sLine := sLine + 'Baselineph_wPO4Ph,';

sLine := sLine + 'Bin,';
sLine := sLine + 'Small_Correct,';
sLine := sLine + 'Num_Proxies,';

// get the variable names for database variables for column headings
for cv in LCRBaseCostSteps.CostVars.Values do
begin
  vname := cv.fID;
  curval := cv.fCurValue;
  rawval := cv.fRawValue;
  sLine := sLine + vname + ',';
  if cv.fImAProb then
  begin
    sLine := sLine + vname + '_r,';
  end;
end;

// get additional variable names for column headings

for j := 4 to Config.YearsOfAnalysis do
begin
  vname := 'pp_lsl_replacement_' + j.ToString;
  sLine := sLine + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_source_chng_' + j.ToString;
  sLine := sLine + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_source_sig_' + j.ToString;
  sLine := sLine + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_treat_change_' + j.ToString;
  sLine := sLine + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'BinChg_' + j.ToString;
  sLine := sLine + vname + ',';
```

```
      end;

      sLine := sLine + 'bBin,';

      sLine := Copy(sLine, 1, Length(sLine)-1);
      Writer.WriteLine(sLine);
    end
    // end column names
    else
    begin
      // read the data lines from sample file
      // replicate rows according the replication factor for particular Type, Size,
Source
      for iRep := 1 to getRepFactor(SystemType, SizeCatSort, SrcWater) do
      begin
        if Config.UseSavedLSLs = 0 then
        begin
          // has_lsl = False
          LCRBaseCostSteps.DetermineSystemPValues(SizeCatSort, SrcWater, 1, CCT,
SystemType, True, 'OH2504412', True, False);
          p_lsl := LCRBaseCostSteps.PWS_p_values.p_lsl;
          if SizeCatSort = 9 then
          begin
            MakeVLSEndPointsLSL(pwsid, LCRBaseCostSteps.PWS_p_values)
          end;
        end
        else
        begin
          tPWSId := PWSId + '_' + iRep.ToString;
          bFound := false;
          bFound := cdsSavedLSLs.FindKey([tPWSId]);
          if bFound then
          begin
            p_lsl := cdsSavedLSLs.FieldByName('LSL').AsInteger;
            LCRBaseCostSteps.PWS_p_values.p_lsl := p_lsl;
            LCRBaseCostSteps.DetermineSystemPValues(SizeCatSort, SrcWater, 1, CCT,
SystemType, True, 'OH2504412', True, True);
          end
          else
            raise Exception.Create('LSL Value not found for: ' + PWSId);
        end;

        //csl('PWSId: ' + PWSId + ', p_lsl: ' + p_lsl.ToString + ', pws_lsl: ' +
LCRBaseCostSteps.PWS_p_values.pws_lsl.ToString);
        bBin := AssignBaselineBin(LCRBaseCostSteps.PWS_p_values,
LCRBaseCostSteps.PWS_p_values.pws_lsl, 'LCR');
        if bBin = 1 then Bin := 1
        else if (bBin >= 2) and (bBin <= 3) then Bin := 2
        else Bin := 3;
```

```
        if Bin = 3 then LCR_P90 := 3
        else if Bin = 2 then LCR_P90 := 12
        else LCR_P90 := 20;

        LCRBaseCostSteps.ResolveDatabaseVariables(SizeCatSort, SrcWater,
LCRBaseCostSteps.PWS_p_values.pws_lsl, CCT, SystemType, PWS90PctBp1, PWS90PctBp2);

        num_lsl := min(population, num_connect) *
LCRBaseCostSteps.PWS_p_values.perc_lsl * LCRBaseCostSteps.PWS_p_values.pws_lsl;

        if CCT = 1 then
        begin
          pbasepo4 := LCRBaseCostSteps.PWS_p_values.pbasepo4;
          pbaseph := LCRBaseCostSteps.PWS_p_values.pbaseph;
          pbasephpo4 := LCRBaseCostSteps.PWS_p_values.pbasephpo4;
        end
        else
        begin
          pbasepo4 := 0;
          pbaseph := 0;
          pbasephpo4 := 0;
        end;

        baselinepo4dose := LCRBaseCostSteps.PWS_p_values.baselinepo4dose;
        baselineph_wph := LCRBaseCostSteps.PWS_p_values.baselineph_wph;
        baselineph_woph := LCRBaseCostSteps.PWS_p_values.baselineph_woph;
        baselineph_wocct := LCRBaseCostSteps.PWS_p_values.baselineph_wocct;
        baselineph_wpo4ph := LCRBaseCostSteps.PWS_p_values.baselineph_wpo4ph;

        connections := min(population, connections);
        sLine1 := '';
        sLine2 := '';
        sLine3 := '';
        sLine1 := sLine1 + xlsIn.GetStringFromCell(r, 1) + '_' + iRep.ToString;
        sLine2 := sLine2 + SystemType.ToString + ',';
        sLine2 := sLine2 + population.ToString + ',';
        sLine2 := sLine2 + num_connect.ToString + ',';
        // 12: Source water
        if GwSw = 'GW' then
          sLine2 := sLine2 + 'Groundwater' + ','
        else if GwSw = 'SW' then
          sLine2 := sLine2 + 'Surface water' + ',';

        sLine2 := sLine2 + CCT.ToString + ',';
        sLine2 := sLine2 + SizeCatSort.ToString + ',';
        // 27: System Location
        sLine2 := sLine2 + xlsIn.GetStringFromCell(r, 27) + ',';
        // 19 Owner Type
        sLine2 := sLine2 + xlsIn.GetStringFromCell(r, 19) + ',';
        sLine2 := sLine2 + first_ale.ToString + ',';
```

```
        sLine2 := sLine2 + getWeight(SystemType, SizeCatSort, SrcWater).ToString +
','';

        if SizeCatSort < 9 then
        begin
          if SystemType = 1 then
          begin
            bFound := false;
            bFound := cdsPFAS_SDWIS.FindKey([PWSId]);
            if bFound then
              numeps := cdsPFAS_SDWIS.FieldByName('EntryPoints').AsInteger
            else
              numeps := getEP(SizeCatSort-1, SrcWater-1);
          end
          else
            numeps := 1;
          sLine2 := sLine2 + numeps.ToString + ',';
        end
        else
        begin
          VLSSystemData.GetSystemData(PWSId);
          numeps := VLSSystemData.Num_EP;
          sLine2 := sLine2 + numeps.ToString + ',';
        end;

        if UseSavedBins then
        begin
          SLSavedData := PWSSampleBins.GetData(xlsIn.GetStringFromCell(r, 1) + '_'
+ iRep.ToString);
          sLine2 := sLine2 + PWSSampleBins.GetDataValue('LSL',SLSavedData) + ',';
        end
        else
          sLine2 := sLine2 + p_lsl.toString + ',';

        sLine2 := sLine2 + num_lsl.ToString + ',';
        sLine2 := sLine2 + pbasepo4.ToString + ',';
        sLine2 := sLine2 + pbaseph.ToString + ',';
        sLine2 := sLine2 + pbasephpo4.ToString + ',';
        sLine2 := sLine2 + baselinepo4dose.ToString + ',';
        sLine2 := sLine2 + baselineph_wph.ToString + ',';
        sLine2 := sLine2 + baselineph_woph.ToString + ',';
        sLine2 := sLine2 + baselineph_wocct.ToString + ',';
        sLine2 := sLine2 + baselineph_wpo4ph.ToString + ',';

        for cv in LCRBaseCostSteps.CostVars.Values do
        begin
          vname := cv.fID;
          curval := cv.fCurValue;
          rawval := cv.fRawValue;
          sLine3 := sLine3 + curVal.ToString + ',';
```

```
            if cv.fImAProb then
            begin
              sLine3 := sLine3 + rawVal.ToString + ',';
            end;
          end;

          SetLength(RepRates, Config.YearsOfAnalysis + 1);
          SetLength(SrcRepRates, Config.YearsOfAnalysis + 1);
          SetLength(SigRepRates, Config.YearsOfAnalysis + 1);
          SetLength(TrtRepRates, Config.YearsOfAnalysis + 1);
          SetLength(LSLVolPctRates, Config.YearsOfAnalysis + 1);
          for j := Low(RepRates) to High(RepRates) do RepRates[j] := 0;

          if UseSavedBins then
          begin
            for j := 4 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('pp_lsl_replacement_' +
j.ToString, SLSavedData) + ',';

            for j := 1 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('p_source_chng_' +
j.ToString, SLSavedData) + ',';

            for j := 1 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('p_source_sig_' +
j.ToString, SLSavedData) + ',';

            for j := 1 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('p_treat_change_' +
j.ToString, SLSavedData) + ',';

            for j := 1 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('BinChg_' + j.ToString,
SLSavedData) + ',';
          end
          else
          begin
            LCRBaseCostSteps.CostVars.DrawLSLReplacementRates(SizeCatSort, SrcWater,

LCRBaseCostSteps.PWS_p_values.pws_lsl,
                                                              CCT, SystemType,
Config.YearsOfAnalysis, 'Baseline',
                                                              RepRates);

            for j := 4 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + RepRates[j].ToString + ',';

            for j := Low(SrcRepRates) to High(SrcRepRates) do SrcRepRates[j] := 0;
            LCRBaseCostSteps.CostVars.DrawP_Source_Chng(SizeCatSort, SrcWater,
```

```
LCRBaseCostSteps.PWS_p_values.pws_lsl,
                                                    CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', SrcRepRates);

          for j := Low(SigRepRates) to High(SigRepRates) do SigRepRates[j] := 0;
          LCRBaseCostSteps.CostVars.DrawP_Source_Sig(SizeCatSort, SrcWater,

LCRBaseCostSteps.PWS_p_values.pws_lsl,
                                                    CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', SigRepRates);

          for j := 1 to Config.YearsOfAnalysis do
            sLine3 := sLine3 + SrcRepRates[j].ToString + ',';

          for j := 1 to Config.YearsOfAnalysis do
            sLine3 := sLine3 + SigRepRates[j].ToString + ',';

          for j := Low(TrtRepRates) to High(TrtRepRates) do TrtRepRates[j] := 0;
          LCRBaseCostSteps.CostVars.DrawP_Treat_Change(SizeCatSort, SrcWater,

LCRBaseCostSteps.PWS_p_values.pws_lsl,
                                                     CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', TrtRepRates);

          for j := 1 to Config.YearsOfAnalysis do
            sLine3 := sLine3 + TrtRepRates[j].ToString + ',';

          for j := 1 to Config.YearsOfAnalysis do
          begin
            BinChg := 0;
            if (SrcRepRates[j] = 1) or (TrtRepRates[j] = 1) then
            begin
              BinChg := LCRBaseCostSteps.AssignNewBinBaseline;
            end;
            sLine3 := sLine3 + BinChg.ToString + ',';
          end;

          // bBin has the values 1 - 5
          sLine3 := sLine3 + bBin.ToString + ',';
        end;

        // This is the LCR Baseline so do not write proy records
        WriteLine(sLine1,sLine2,sLine3,Bin,0,0,'');
      end;
    end;
  end;

  Writer.Free;
  XlsIn.Free;
```

```
  SLVlsLSLs.SaveToFile(ExtractFilePath(SDWISFile) + 'VLSBaseline_LSL.csv')
end;

procedure TPWSReplicator.WriteLCRRBaseline;
var
  Reader: TStreamReader;
  Writer: TStreamWriter;
  inFileName, outFileName: string;
  sLineIn, sLineOut: string;

  slInData, slInCols: TStringList;
  SystemType, SizeCat, SrcWater: integer;
  population: integer;
  LSL, CCT: integer;

  cv: TCostVar;
  vname: string;
  curVal, rawVal: double;

  r, j: integer;
  rr: double;

  RepRates: TDoubleArray;
  LSLReplacementRates: TDoubleArray;
  LSLVolPctRates: TDoubleArray;
  arrBinChg: TDoubleArray;

  Bin: integer;
  nn1, nn2: string;
  SLSavedData: TStringList;

  bBin, BaseBin: integer;
  LCRx_Bin_Initial, LCRx_Bin, LCRx_P90: integer;
  LCRx_Bin_15, LCRx_Bin_12, LCRx_Bin_10, LCRx_Bin_5: integer;
  bALENotAchieved: integer;

  pws_lsl: integer;
  pwsid, hpwsid, sproxy: string;

  procedure WriteLine(Bin, SmallCorrect, NumProxies: integer; suffix: string);
  var
    cv2: TCostVar;
    j2: integer;
    vname2: string;
    CCT, LSL: integer;
    SystemType: integer;
    xBin: string;
    pwsid, sproxy: string;
  begin
    // write the data values for the sample file
```

```
    SystemType := slInData.Strings[slInCols.IndexOf('SystemType')].ToInteger;
    CCT := slInData.Strings[slInCols.IndexOf('CCT')].ToInteger;
    LSL := slInData.Strings[slInCols.IndexOf('LSL')].ToInteger;

    pwsid := slInData.Strings[slInCols.IndexOf('PWSID')];
    sProxy := RightStr(pwsid,1);
    if (sProxy = 'a') or (sProxy = 'b') or (sProxy = 'c') then
      pwsid :=  Copy(pwsid, 1, Length(pwsid)-1);

    sLineOut := '';
    sLineOut := sLineOut + pwsid + suffix + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('SystemType')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Population')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Connections')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('SourceWater')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCT')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('PopCat')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('State')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Owner')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('First_ale')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Weight')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('NumberEPs')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('LSL')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('NumberLSLs')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCTPO4')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCTPH')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCTBoth')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('BaselinePO4Dose')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_wPh')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_woPh')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_woCCT')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_wPO4Ph')] +
',';

    if UseSavedBins then
    begin
      SLSavedData :=
PWSSampleBins.GetData(slInData.Strings[slInCols.IndexOf('PWSID')]);
      xBin := PWSSampleBins.GetDataValue('Bin',SLSavedData);
      if xBin = '' then
        raise Exception.Create('Unable to get bin value for PWSId: ' +
slInData.Strings[slInCols.IndexOf('PWSID')]);
      sLineOut := sLineOut + xBin + ',';
    end
    else
      sLineOut := sLineOut + Bin.ToString + ',';
```

```
      sLineOut := sLineOut + SmallCorrect.ToString + ',';
      sLineOut := sLineOut + NumProxies.ToString + ',';

      for cv2 in LCRRBaseCostSteps.CostVars.Values do
      begin
        vname := cv2.fID;
        curval := cv2.fCurValue;
        rawval := cv2.fRawValue;
        sLineOut := sLineOut + curVal.ToString + ',';
        if cv2.fImAProb then
        begin
          sLineOut := sLineOut + rawVal.ToString + ',';
        end;
      end;

      for j2 := 4 to Config.YearsOfAnalysis do
      begin
        if UseSavedBins then
          sLineOut := sLineOut + PWSSampleBins.GetDataValue('pp_lsl_replacement_' +
j2.ToString, SLSavedData) + ','
        else
          sLineOut := sLineOut + LSLReplacementRates[j2].ToString + ',';
      end;

      for j2 := 1 to Config.YearsOfAnalysis do
      begin
        vname2 := 'p_source_chng_' + j2.ToString;
        sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf(vname2)] + ',';
      end;

      for j2 := 1 to Config.YearsOfAnalysis do
      begin
        vname2 := 'p_source_sig_' + j2.ToString;
        sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf(vname2)] + ',';
      end;

      for j2 := 1 to Config.YearsOfAnalysis do
      begin
        vname2 := 'p_treat_change_' + j2.ToString;
        sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf(vname2)] + ',';
      end;

      for j2 := 1 to Config.YearsOfAnalysis do
      begin
        if UseSavedBins then
          sLineOut := sLineOut +
PWSSampleBins.GetDataValue('pp_lsl_replaced_vol_pct' + j2.ToString, SLSavedData) +
','
        else
```

```
          sLineOut := sLineOut + LSLVolPctRates[j2].ToString + ',';
      end;

      for j2 := 1 to Config.YearsOfAnalysis do
      begin
        if UseSavedBins then
          sLineOut := sLineOut + PWSSampleBins.GetDataValue('BinChg_' + j2.ToString,
SLSavedData) + ','
        else
          sLineOut := sLineOut + arrBinChg[j2].ToString + ',';
      end;

      // bBin has values 1 - 5
      sLineOut := sLineOut + bBin.ToString + ',';
      sLineOut := sLineOut + BaseBin.ToString + ',';
      sLineOut := Copy(sLineOut, 1, Length(sLineOut)-1);
      Writer.WriteLine(sLineOut);
  end;
begin
  // Write the LCRR baseline based on the LCR baseline

  if UseSavedBins then
  begin
    PWSSampleBins.ReadBinFile(false, LSLLevel, SmallProxyLabel);
  end;

  inFileName := ExtractFilePath(SDWISFile) + 'LCRBaseline_' + SampleName + '.csv';
  Reader := TStreamReader.Create(inFileName);

  outFileName := ExtractFilePath(SDWISFile) + 'LCRRBaseline_' + SampleName + '.csv';
  OptionSampleFilename := outFileName;
  Writer := TStreamWriter.Create(outFileName);

  slInData := TStringList.Create;
  slInData.Delimiter := ',';
  slInData.StrictDelimiter := true;
  slInCols := TStringList.Create;
  slInCols.Delimiter := ',';
  slInCols.StrictDelimiter := true;

  // write column headings for outFile
       sLineOut := '';
       sLineOut := sLineOut + 'PWSID,';
       sLineOut := sLineOut + 'SystemType,';
       sLineOut := sLineOut + 'Population,';
       sLineOut := sLineOut + 'Connections,';
       sLineOut := sLineOut + 'SourceWater,';
       sLineOut := sLineOut + 'CCT,';
       sLineOut := sLineOut + 'PopCat,';
       sLineOut := sLineOut + 'State,';
```

```
sLineOut := sLineOut + 'Owner,';
sLineOut := sLineOut + 'First_ale,';
sLineOut := sLineOut + 'Weight,';
sLineOut := sLineOut + 'NumberEPs,';
sLineOut := sLineOut + 'LSL,';
sLineOut := sLineOut + 'NumberLSLs,';
sLineOut := sLineOut + 'CCTPO4,';
sLineOut := sLineOut + 'CCTPH,';
sLineOut := sLineOut + 'CCTBoth,';
sLineOut := sLineOut + 'BaselinePO4Dose,';
sLineOut := sLineOut + 'Baselineph_wPh,';
sLineOut := sLineOut + 'Baselineph_woPh,';
sLineOut := sLineOut + 'Baselineph_woCCT,';
sLineOut := sLineOut + 'Baselineph_wPO4Ph,';

sLineOut := sLineOut + 'Bin,';
sLineOut := sLineOut + 'Small_Correct,';
sLineOut := sLineOut + 'Num_Proxies,';

for cv in LCRRBaseCostSteps.CostVars.Values do
begin
  vname := cv.fID;
  sLineOut := sLineOut + vname + ',';
  if cv.fImAProb then
  begin
    sLineOut := sLineOut + vname + '_r,';
  end;
end;

for j := 4 to Config.YearsOfAnalysis do
begin
  vname := 'pp_lsl_replacement_' + j.ToString;
  sLineOut := sLineOut + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_source_chng_' + j.ToString;
  sLineOut := sLineOut + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_source_sig_' + j.ToString;
  sLineOut := sLineOut + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_treat_change_' + j.ToString;
```

```
      sLineOut := sLineOut + vname + ',';
    end;

    for j := 1 to Config.YearsOfAnalysis do
    begin
      vname := 'pp_lsl_replaced_vol_pct_' + j.ToString;
      sLineOut := sLineOut + vname + ',';
    end;

    for j := 1 to Config.YearsOfAnalysis do
    begin
      vname := 'BinChg_' + j.ToString;
      sLineOut := sLineOut + vname + ',';
    end;

    sLineOut := sLineOut + 'bBin,';
    sLineOut := sLineOut + 'BaseBin,';
    sLineOut := Copy(sLineOut, 1, Length(sLineOut)-1);
    Writer.WriteLine(sLineOut);

// read first row of inFile to get column headings
sLineIn := Reader.ReadLine;
slInCols.CommaText := sLineIn;

hpwsid := '';
r := 1;
while not Reader.EndOfStream do
begin
  sLineIn := Reader.ReadLine;

  slInData.CommaText := sLineIn;

  pwsid := slInData.Strings[slInCols.IndexOf('PWSID')];
  sProxy := RightStr(pwsid,1);
  if (sProxy = 'a') or (sProxy = 'b') or (sProxy = 'c') then
    pwsid :=  Copy(pwsid, 1, Length(pwsid)-1);

  if hpwsid = pwsid then continue;
  hpwsid := pwsid;

  if slInData.Strings[slInCols.IndexOf('SourceWater')] = 'Groundwater' then
    SrcWater := 1
  else
    SrcWater := 2;

  SystemType := slInData.Strings[slInCols.IndexOf('SystemType')].ToInteger;
  SizeCat := slInData.Strings[slInCols.IndexOf('PopCat')].ToInteger;

  population := slInData.Strings[slInCols.IndexOf('Population')].ToInteger;
```

```
    LSL := slInData.Strings[slInCols.IndexOf('LSL')].ToInteger;
    if LSL > 0 then
      pws_lsl := 1
    else
      pws_lsl := 0;

    CCT := slInData.Strings[slInCols.IndexOf('CCT')].ToInteger;

    bBin := slInData.Strings[slInCols.IndexOf('bBin')].ToInteger;
    baseBin := bBin;
    LCRRBaseCostSteps.DetermineSystemPValues(SizeCat, SrcWater, pws_lsl, CCT,
SystemType, False, 'OH2504412', True, True);

    bBin := AssignOptionBin2(pws_lsl, LCRRBaseCostSteps.PWS_p_values, bBin,
OptionName);

    if bBin = 1 then Bin := 1
    else if (bBin = 2) or (bBin = 3) then Bin := 2
    else Bin := 3;

    if Bin = 3 then LCRx_P90 := 3
    else if Bin = 2 then LCRx_P90 := 12
    else LCRx_P90 := 20;

    LCRRBaseCostSteps.ResolveDatabaseVariables(SizeCat, SrcWater, pws_lsl, CCT,
SystemType,
                                           PWS90PctBp1, PWS90PctBp2, slInCols,
slInData);

    if not UseSavedBins then
    begin
      rr := iiRandom(rrBase);
      if rr < 0.15 then bALENotAchieved := 1
      else bALENotAchieved := 0;

      SetLength(LSLReplacementRates, Config.YearsOfAnalysis + 1);
      for j := Low(LSLReplacementRates) to High(LSLReplacementRates) do
LSLReplacementRates[j] := 0;
      LCRRBaseCostSteps.CostVars.DrawLSLReplacementRates(SizeCat, SrcWater,
                                                  pws_lsl, CCT, SystemType,
Config.YearsOfAnalysis, OptionName,
                                                  LSLReplacementRates);

      SetLength(LSLVolPctRates, Config.YearsOfAnalysis + 1);
      for j := Low(LSLVolPctRates) to High(LSLVolPctRates) do LSLVolPctRates[j] :=
0;
      LCRRBaseCostSteps.CostVars.DrawPp_LSL_Replaced_Vol_Pct(SizeCat, SrcWater,
                                                     pws_lsl, CCT, SystemType,
Config.YearsOfAnalysis, '', LSLVolPctRates);
```

```
      SetLength(arrBinChg, Config.YearsOfAnalysis + 1);

      for j := 1 to Config.YearsOfAnalysis do
      begin
        nn1 := 'p_source_chng_' + j.ToString;
        nn2 := 'p_treat_change_' + j.ToString;
        arrBinChg[j] := 0;
        if (slInData.Strings[slInCols.IndexOf(nn1)] = '1') or
(slInData.Strings[slInCols.IndexOf(nn2)] = '1') then
        begin
          arrBinChg[j] := LCRRBaseCostSteps.AssignNewBinOption(OptionName);
        end;
      end;
    end;

{
  for small systems Pop <= 3300
  LSL   CCT     Small_Correct   Num_Proxies
  0       0,1     2,3                   2
  1       0,1     1,2,3               3

  for other systems Num_Proxies = 0
}
     if ProxyRecords then begin
       if (SystemType = 1) and (population > SmallProxyPop) then
         WriteLine(Bin,0,0,'')
       else begin
         if (pws_lsl = 0) then begin
           WriteLine(Bin,2,2,'b');
           WriteLine(Bin,3,2,'c');
         end
         else if (pws_lsl = 1)then begin
           WriteLine(Bin,1,3,'a');
           WriteLine(Bin,2,3,'b');
           WriteLine(Bin,3,3,'c');
         end
       end;
     end else
       WriteLine(Bin,0,0,'');

    inc(r);
    //if r > 10 then break;
  end;

  Reader.Free;
  Writer.Free;

  slInData.Free;
  slInCols.Free;
end;
```

```
procedure TPWSReplicator.WriteBaseline3;
var
  XlsIn: TExcelFile;
  r: integer;

  SizeCatSort: integer;
  GwSw: string;
  PWSType: string;
  SrcWater: integer;
  PWSId: string;
  CCT: integer;
  population: integer;
  connections: integer;
  first_ale: integer;
  numeps: integer;
  SystemType: integer;

  p_lsl: integer;
  num_lsl: double;
  pbasepo4, pbaseph, pbasephpo4, baselinepo4dose: integer;
  baselineph_wph, baselineph_woph, baselineph_wocct, baselineph_wpo4ph: integer;

  num_pop_connection: double;
  correct_num_pop_connection: double;
  num_connect: integer;
  tmpstr: string;
  tmpfloat: double;

  Outfile: string;

  cv: TCostVar;
  vname: string;
  curval, rawval: double;
  rCol, j, k, iRep: integer;

  Writer: TStreamWriter;
  sLine, sLine1, sLine2, sLine3: string;

  RepRates: TDoubleArray;
  SrcRepRates: TDoubleArray;
  SigRepRates: TDoubleArray;
  TrtRepRates: TDoubleArray;
  LSLVolPctRates: TDoubleArray;

  Bin: integer;
  BinChg: integer;

  bBin: integer;
  LCR_Bin, LCR_P90: integer;
```

```
  SLSavedData: TStringList;
  bFound: boolean;
  tPWSId: string;

  procedure WriteLine(sLine1, sLine2, sLine3: string; Bin, SmallCorrect, NumProxies:
integer; suffix: string);
  var
    sLineOut: string;
  begin

    sLineOut := sLine1 + suffix + ',';
    sLineOut := sLineOut + sLine2;
    sLineOut := sLineOut + Bin.ToString + ',';
    sLineOut := sLineOut + SmallCorrect.ToString + ',';
    sLineOut := sLineOut + NumProxies.ToString + ',';
    sLineOut := sLineOut + sLine3;

    sLineOut := Copy(sLineOut, 1, Length(sLineOut)-1);
    Writer.WriteLine(sLineOut);
  end;
begin
  // This is for the LCRR baseline.

  if UseSavedBins then
  begin
    PWSSampleBins.ReadBinFile(true, LSLLevel, SmallProxyLabel);
  end;

  // original SDWIS sample file
  XlsIn := TXlsFile.Create(SDWISFile, False);
  XlsIn.ActiveSheetByName := 'Inventory Characteristics';

  OutFile := ExtractFilePath(SDWISFile) + 'Baseline_' + SampleName + '.csv';
  BaselineSampleFilename := OutFile;
  Writer := TStreamWriter.Create(OutFile);

  SystemType := 0;
  SizeCatSort := 0;
  SrcWater := 0;
  connections := 0;
  numeps := 0;

  cdsPFAS_SDWIS.IndexName := 'IdxAll';
  if Config.UseSavedLSLs = 1 then
    cdsSavedLSLs.IndexName := 'IdxAll';
  cdsMissingSrcWater.IndexName := 'IdxAll';

  for r := 1 to XlsIn.RowCount do begin
     // skip certain rows from the SDWIS sample for different reasons
```

```
if r > 1 then
begin
  if MakeProfileSample then begin
    if Random>SampRate then continue;
  end;

  SizeCatSort := XlsIn.GetCellValue(r, 13).AsVariant;

  PWSId := XlsIn.GetStringFromCell(r, 1);

  // Exclude PWS JOINT REGIONAL WATER SUPPLY SYSTEM CA
  if PWSId = 'CA3010071' then
    continue;

  // exclude invalid GW or SW Code
  GwSw := xlsIn.GetStringFromCell(r, 12);

  if GwSw = 'Groundwater' then
    GwSw := 'GW' else
  if GwSw = 'Surface water' then
    GwSw := 'SW';

  if GwSw = '-' then
  begin
    bFound := false;
    bFound := cdsMissingSrcWater.FindKey([PWSId]);
    if bFound then
      GwSw := cdsMissingSrcWater.FieldByName('PrimarySource').AsString
    else
      raise Exception.Create('Primary water source not found for: ' + PWSId)
  end;

  if GwSW = 'GW' then SrcWater := 1
  else SrcWater := 2;

  PWSType := xlsIn.GetStringFromCell(r, 3);

  if PWSType = 'Community water system' then
    PWSType := 'CWS' else
  if PWSType = 'Non-Transient non-community system' then
    PWSType := 'NTNCWS';

  if PWSType = 'CWS' then
    SystemType := 1
  else
    SystemType := 2;

  if xlsIn.GetStringFromCell(r, 38) = 'FALSE' then
    CCT := 0
  else
```

```
        CCT := 1;

      first_ale := 0;

      tmpstr := xlsIn.GetCellValue(r, 7).AsVariant;
      population := tmpstr.Replace(',','', [rfReplaceAll,
rfIgnoreCase]).ToInteger();
      if population < 25 then population := 25;

      tmpstr := xlsIn.GetCellValue(r, 10).AsVariant;
      tmpfloat := tmpstr.Replace(',','', [rfReplaceAll, rfIgnoreCase]).ToDouble();
      connections := round(tmpfloat);

      if connections > 0 then
        num_pop_connection := population / connections
      else
        num_pop_connection := 0;

      if (num_pop_connection >= 1) and (num_pop_connection <= 5) then
        correct_num_pop_connection := num_pop_connection
      else
        correct_num_pop_connection := CatConnectionMean[SystemType, SizeCatSort,
SrcWater];

      if correct_num_pop_connection > 0 then
        num_connect := round(population / correct_num_pop_connection)
      else
        num_connect := 1;

      if SystemType = 2 then
      begin
        num_connect := connections;
        if num_connect >= 20 then
          num_connect := 20;
      end;
    end;  // end skip some rows...

    // read the column names from sample file
    if r = 1 then
    begin
      sLine := '';
      sLine := sLine + 'PWSID,';
      sLine := sLine + 'SystemType,';
      sLine := sLine + 'Population,';
      sLine := sLine + 'Connections,';
      sLine := sLine + 'SourceWater,';
      sLine := sLine + 'CCT,';
      sLine := sLine + 'PopCat,';
      sLine := sLine + 'State,';
      sLine := sLine + 'Owner,';
```

```
sLine := sLine + 'First_ale,';
sLine := sLine + 'Weight,';
sLine := sLine + 'NumberEPs,';
sLine := sLine + 'LSL,';
sLine := sLine + 'NumberLSLs,';
sLine := sLine + 'CCTPO4,';
sLine := sLine + 'CCTPH,';
sLine := sLine + 'CCTBoth,';
sLine := sLine + 'BaselinePO4Dose,';
sLine := sLine + 'Baselineph_wPh,';
sLine := sLine + 'Baselineph_woPh,';
sLine := sLine + 'Baselineph_woCCT,';
sLine := sLine + 'Baselineph_wPO4Ph,';

sLine := sLine + 'Bin,';
sLine := sLine + 'Small_Correct,';
sLine := sLine + 'Num_Proxies,';

for cv in BaseCostSteps.CostVars.Values do
begin
  vname := cv.fID;
  curval := cv.fCurValue;
  rawval := cv.fRawValue;
  sLine := sLine + vname + ',';
  if cv.fImAProb then
  begin
    sLine := sLine + vname + '_r,';
  end;
end;

for j := 4 to Config.YearsOfAnalysis do
begin
  vname := 'pp_lsl_replacement_' + j.ToString;
  sLine := sLine + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_source_chng_' + j.ToString;
  sLine := sLine + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
  vname := 'p_source_sig_' + j.ToString;
  sLine := sLine + vname + ',';
end;

for j := 1 to Config.YearsOfAnalysis do
begin
```

```
        vname := 'p_treat_change_' + j.ToString;
        sLine := sLine + vname + ',';
      end;

      for j := 1 to Config.YearsOfAnalysis do
      begin
        vname := 'BinChg_' + j.ToString;
        sLine := sLine + vname + ',';
      end;

      sLine := sLine + 'bBin,';

      if BaselineName = 'LCRR' then
      begin
        for j := 1 to Config.YearsOfAnalysis do
        begin
          vname := 'pp_lsl_replaced_vol_pct_' + j.ToString;
          sLine := sLine + vname + ',';
        end;
      end;

      sLine := Copy(sLine, 1, Length(sLine)-1);
      Writer.WriteLine(sLine);
    end
    // end column names
    else
    begin
      // read the data lines from sample file
      for iRep := 1 to getRepFactor(SystemType, SizeCatSort, SrcWater) do
      begin
        if Config.UseSavedLSLs = 0 then
        begin
          BaseCostSteps.DetermineSystemPValues(SizeCatSort, SrcWater, 1, CCT,
SystemType, True, 'OH2504412', True, False);
          p_lsl := BaseCostSteps.PWS_p_values.p_lsl;
        end
        else
        begin
          tPWSId := PWSId + '_' + iRep.ToString;
          bFound := false;
          bFound := cdsSavedLSLs.FindKey([tPWSId]);
          if bFound then
          begin
            p_lsl := cdsSavedLSLs.FieldByName('LSL').AsInteger;
            BaseCostSteps.PWS_p_values.p_lsl := p_lsl;
            BaseCostSteps.DetermineSystemPValues(SizeCatSort, SrcWater, 1, CCT,
SystemType, True, 'OH2504412', True, True);
          end
          else
            raise Exception.Create('LSL Value not found for: ' + PWSId);
```

```
        end;

        if BaselineName = 'LCR' then
        begin
          bBin := AssignBaselineBin(BaseCostSteps.PWS_p_values,
BaseCostSteps.PWS_p_values.pws_lsl, BaselineName);
          if bBin = 1 then Bin := 1
          else if (bBin >= 2) and (bBin <= 3) then Bin := 2
          else Bin := 3;

          if Bin = 3 then LCR_P90 := 3
          else if Bin = 2 then LCR_P90 := 12
          else LCR_P90 := 20;
        end
        else if BaselineName = 'LCRR' then
        begin
          bBin := AssignBaselineBin(BaseCostSteps.PWS_p_values,
BaseCostSteps.PWS_p_values.pws_lsl, BaselineName);

          if bBin = 1 then Bin := 1
          else if (bBin = 2) or (bBin = 3) then Bin := 2
          else Bin := 3;
        end;

        BaseCostSteps.ResolveDatabaseVariables(SizeCatSort, SrcWater,
BaseCostSteps.PWS_p_values.pws_lsl, CCT, SystemType, PWS90PctBp1, PWS90PctBp2);

        num_lsl := min(population, num_connect) *
BaseCostSteps.PWS_p_values.perc_lsl * BaseCostSteps.PWS_p_values.pws_lsl;

        if CCT = 1 then
        begin
          pbasepo4 := BaseCostSteps.PWS_p_values.pbasepo4;
          pbaseph := BaseCostSteps.PWS_p_values.pbaseph;
          pbasephpo4 := BaseCostSteps.PWS_p_values.pbasephpo4;
        end
        else
        begin
          pbasepo4 := 0;
          pbaseph := 0;
          pbasephpo4 := 0;
        end;

        baselinepo4dose := BaseCostSteps.PWS_p_values.baselinepo4dose;
        baselineph_wph := BaseCostSteps.PWS_p_values.baselineph_wph;
        baselineph_woph := BaseCostSteps.PWS_p_values.baselineph_woph;
        baselineph_wocct := BaseCostSteps.PWS_p_values.baselineph_wocct;
        baselineph_wpo4ph := BaseCostSteps.PWS_p_values.baselineph_wpo4ph;

        connections := min(population, connections);
```

```
        sLine1 := '';
        sLine2 := '';
        sLine3 := '';
        sLine1 := sLine1 + xlsIn.GetStringFromCell(r, 1) + '_' + iRep.ToString;
        sLine2 := sLine2 + SystemType.ToString + ',';
        sLine2 := sLine2 + population.ToString + ',';
        sLine2 := sLine2 + num_connect.ToString + ',';
        // 12: Source water
        if GwSw = 'GW' then
          sLine2 := sLine2 + 'Groundwater' + ','
        else if GwSw = 'SW' then
          sLine2 := sLine2 + 'Surface water' + ',';

        sLine2 := sLine2 + CCT.ToString + ',';
        sLine2 := sLine2 + SizeCatSort.ToString + ',';
        // 27: System Location
        sLine2 := sLine2 + xlsIn.GetStringFromCell(r, 27) + ',';
        // 19 Owner Type
        sLine2 := sLine2 + xlsIn.GetStringFromCell(r, 19) + ',';
        sLine2 := sLine2 + first_ale.ToString + ',';
        sLine2 := sLine2 + getWeight(SystemType, SizeCatSort, SrcWater).ToString +
',';

        if PWSId = 'OH2504412' then
        begin
          numeps := 30;
          sLine2 := sLine2 + '30' + ',';
        end
        else if SizeCatSort < 9 then
        begin
          if SystemType = 1 then
          begin
            bFound := false;
            bFound := cdsPFAS_SDWIS.FindKey([PWSId]);
            if bFound then
              numeps := cdsPFAS_SDWIS.FieldByName('EntryPoints').AsInteger
            else
              numeps := getEP(SizeCatSort-1, SrcWater-1);
          end
          else
            numeps := 1;
          sLine2 := sLine2 + numeps.ToString + ',';
        end
        else
        begin
          VLSSystemData.GetSystemData(PWSId);
          numeps := VLSSystemData.Num_EP;
          sLine2 := sLine2 + numeps.ToString + ',';
        end;
```

```
          if UseSavedBins then
            SLSavedData := PWSSampleBins.GetData(xlsIn.GetStringFromCell(r, 1) + '_'
+ iRep.ToString);

          if UseSavedBins then
            sLine2 := sLine2 + PWSSampleBins.GetDataValue('LSL',SLSavedData) + ','
          else
            sLine2 := sLine2 + p_lsl.toString + ',';

          sLine2 := sLine2 + num_lsl.ToString + ',';
          sLine2 := sLine2 + pbasepo4.ToString + ',';
          sLine2 := sLine2 + pbaseph.ToString + ',';
          sLine2 := sLine2 + pbasephpo4.ToString + ',';
          sLine2 := sLine2 + baselinepo4dose.ToString + ',';
          sLine2 := sLine2 + baselineph_wph.ToString + ',';
          sLine2 := sLine2 + baselineph_woph.ToString + ',';
          sLine2 := sLine2 + baselineph_wocct.ToString + ',';
          sLine2 := sLine2 + baselineph_wpo4ph.ToString + ',';

          for cv in BaseCostSteps.CostVars.Values do
          begin
            vname := cv.fID;
            curval := cv.fCurValue;
            rawval := cv.fRawValue;
            sLine3 := sLine3 + curVal.ToString + ',';
            if cv.fImAProb then
            begin
              sLine3 := sLine3 + rawVal.ToString + ',';
            end;
          end;

          SetLength(RepRates, Config.YearsOfAnalysis + 1);
          SetLength(SrcRepRates, Config.YearsOfAnalysis + 1);
          SetLength(SigRepRates, Config.YearsOfAnalysis + 1);
          SetLength(TrtRepRates, Config.YearsOfAnalysis + 1);
          SetLength(LSLVolPctRates, Config.YearsOfAnalysis + 1);
          for j := Low(RepRates) to High(RepRates) do RepRates[j] := 0;

          if UseSavedBins then
          begin
            for j := 4 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('pp_lsl_replacement_' +
j.ToString, SLSavedData) + ',';

            for j := 1 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('p_source_chng_' +
j.ToString, SLSavedData) + ',';

            for j := 1 to Config.YearsOfAnalysis do
              sLine3 := sLine3 + PWSSampleBins.GetDataValue('p_source_sig_' +
```

```
j.ToString, SLSavedData) + ',';

           for j := 1 to Config.YearsOfAnalysis do
             sLine3 := sLine3 + PWSSampleBins.GetDataValue('p_treat_change_' +
j.ToString, SLSavedData) + ',';

           for j := 1 to Config.YearsOfAnalysis do
             sLine3 := sLine3 + PWSSampleBins.GetDataValue('BinChg_' + j.ToString,
SLSavedData) + ',';
         end
         else
         begin
           BaseCostSteps.CostVars.DrawLSLReplacementRates(SizeCatSort, SrcWater,

BaseCostSteps.PWS_p_values.pws_lsl,
                                                          CCT, SystemType,
Config.YearsOfAnalysis, 'Baseline',
                                                          RepRates);

           for j := 4 to Config.YearsOfAnalysis do
             sLine3 := sLine3 + RepRates[j].ToString + ',';

           for j := Low(SrcRepRates) to High(SrcRepRates) do SrcRepRates[j] := 0;
           BaseCostSteps.CostVars.DrawP_Source_Chng(SizeCatSort, SrcWater,

BaseCostSteps.PWS_p_values.pws_lsl,
                                                    CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', SrcRepRates);

           for j := Low(SigRepRates) to High(SigRepRates) do SigRepRates[j] := 0;
           BaseCostSteps.CostVars.DrawP_Source_Sig(SizeCatSort, SrcWater,

BaseCostSteps.PWS_p_values.pws_lsl,
                                                    CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', SigRepRates);

           for j := 1 to Config.YearsOfAnalysis do
             sLine3 := sLine3 + SrcRepRates[j].ToString + ',';

           for j := 1 to Config.YearsOfAnalysis do
             sLine3 := sLine3 + SigRepRates[j].ToString + ',';

           for j := Low(TrtRepRates) to High(TrtRepRates) do TrtRepRates[j] := 0;
           BaseCostSteps.CostVars.DrawP_Treat_Change(SizeCatSort, SrcWater,

BaseCostSteps.PWS_p_values.pws_lsl,
                                                     CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', TrtRepRates);

           for j := 1 to Config.YearsOfAnalysis do
```

```
            sLine3 := sLine3 + TrtRepRates[j].ToString + ',';

           for j := 1 to Config.YearsOfAnalysis do
           begin
             BinChg := 0;
             if (SrcRepRates[j] = 1) or (TrtRepRates[j] = 1) then
             begin
               BinChg := BaseCostSteps.AssignNewBinBaseline;
             end;
             sLine3 := sLine3 + BinChg.ToString + ',';
           end;

           // bBin has the values 1 - 5
           sLine3 := sLine3 + bBin.ToString + ',';


          for j := Low(LSLVolPctRates) to High(LSLVolPctRates) do LSLVolPctRates[j]
:= 0;
          BaseCostSteps.CostVars.DrawPp_LSL_Replaced_Vol_Pct(SizeCatSort, SrcWater,

BaseCostSteps.PWS_p_values.pws_lsl,
                                                    CCT, SystemType,
Config.YearsOfAnalysis, '', LSLVolPctRates);
           for j := 1 to Config.YearsOfAnalysis do
           begin
             sLine3 := sLine3 + LSLVolPctRates[j].ToString + ',';
           end;
         end;

         {
          for small systems Pop <= 3300
          LSL   CCT     Small_Correct   Num_Proxies
          0       0,1     2,3                   2
          1       0,1     1,2,3               3

          for other systems Num_Proxies = 0
         }
         if ProxyRecords then begin
           if (SystemType = 1) and (population > SmallProxyPop) then
             WriteLine(sLine1,sLine2,sLine3,Bin,0,0,'')
           else begin
             if (p_lsl = 0) then begin
               WriteLine(sLine1,sLine2,sLine3,Bin,2,2,'b');
               WriteLine(sLine1,sLine2,sLine3,Bin,3,2,'c');
             end
             else if (p_lsl > 0)then begin
               WriteLine(sLine1,sLine2,sLine3,Bin,1,3,'a');
               WriteLine(sLine1,sLine2,sLine3,Bin,2,3,'b');
               WriteLine(sLine1,sLine2,sLine3,Bin,3,3,'c');
             end
```

```
          end;
        end else
          WriteLine(sLine1,sLine2,sLine3,Bin,0,0,'');
      end;
    end;
 end;

  Writer.Free;
  XlsIn.Free;
end;

procedure TPWSReplicator.WriteOption;
var
  Reader: TStreamReader;
  Writer: TStreamWriter;
  inFileName, outFileName: string;
  sLineIn, sLineOut: string;

  slInData, slInCols: TStringList;
  SystemType, SizeCat, SrcWater: integer;
  population: integer;
  LSL, CCT: integer;

  cv: TCostVar;
  vname: string;
  curVal, rawVal: double;

  r, j: integer;
  rr: double;

  RepRates: TDoubleArray;
  LSLReplacementRates: TDoubleArray;
  LSLVolPctRates: TDoubleArray;
  arrBinChg: TDoubleArray;

  Bin: integer;
  nn1, nn2: string;
  SLSavedData: TStringList;

  bBin, BaseBin: integer;
  LCRx_Bin_Initial, LCRx_Bin, LCRx_P90: integer;
  LCRx_Bin_15, LCRx_Bin_12, LCRx_Bin_10, LCRx_Bin_5: integer;
  bALENotAchieved: integer;

  pws_lsl: integer;
  pwsid, hpwsid, sproxy: string;

  procedure WriteLine(Bin, SmallCorrect, NumProxies: integer; suffix: string);
  var
    cv2: TCostVar;
```

```
    j2: integer;
    vname2: string;
    CCT, LSL: integer;
    SystemType: integer;
    xBin: string;
    pwsid, sproxy: string;
  begin
    SystemType := slInData.Strings[slInCols.IndexOf('SystemType')].ToInteger;
    CCT := slInData.Strings[slInCols.IndexOf('CCT')].ToInteger;
    LSL := slInData.Strings[slInCols.IndexOf('LSL')].ToInteger;

    pwsid := slInData.Strings[slInCols.IndexOf('PWSID')];
    sProxy := RightStr(pwsid,1);
    if (sProxy = 'a') or (sProxy = 'b') or (sProxy = 'c') then
      pwsid :=  Copy(pwsid, 1, Length(pwsid)-1);

    sLineOut := '';
    sLineOut := sLineOut + pwsid + suffix + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('SystemType')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Population')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Connections')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('SourceWater')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCT')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('PopCat')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('State')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Owner')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('First_ale')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Weight')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('NumberEPs')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('LSL')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('NumberLSLs')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCTPO4')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCTPH')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('CCTBoth')] + ',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('BaselinePO4Dose')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_wPh')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_woPh')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_woCCT')] +
',';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('Baselineph_wPO4Ph')] +
',';

        if UseSavedBins then
          SLSavedData :=
PWSSampleBins.GetData(slInData.Strings[slInCols.IndexOf('PWSID')]);

        if UseSavedBins then
```

```
          begin
            xBin := PWSSampleBins.GetDataValue('Bin',SLSavedData);
            if xBin = '' then
              raise Exception.Create('Unable to get bin value for PWSId: ' +
slInData.Strings[slInCols.IndexOf('PWSID')]);
            sLineOut := sLineOut + xBin + ',';
          end
          else
            sLineOut := sLineOut + Bin.ToString + ',';

          sLineOut := sLineOut + SmallCorrect.ToString + ',';
          sLineOut := sLineOut + NumProxies.ToString + ',';

          for cv2 in ScenCostSteps.CostVars.Values do
          begin
            vname := cv2.fID;
            curval := cv2.fCurValue;
            rawval := cv2.fRawValue;
            sLineOut := sLineOut + curVal.ToString + ',';
            if cv2.fImAProb then
            begin
              sLineOut := sLineOut + rawVal.ToString + ',';
            end;
          end;

          for j2 := 4 to Config.YearsOfAnalysis do
          begin
            if UseSavedBins then
              sLineOut := sLineOut + PWSSampleBins.GetDataValue('pp_lsl_replacement_'
+ j2.ToString, SLSavedData) + ','
            else
              sLineOut := sLineOut + LSLReplacementRates[j2].ToString + ',';
          end;

          for j2 := 1 to Config.YearsOfAnalysis do
          begin
            vname2 := 'p_source_chng_' + j2.ToString;
            sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf(vname2)] + ',';
          end;

          for j2 := 1 to Config.YearsOfAnalysis do
          begin
            vname2 := 'p_source_sig_' + j2.ToString;
            sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf(vname2)] + ',';
          end;

          for j2 := 1 to Config.YearsOfAnalysis do
          begin
            vname2 := 'p_treat_change_' + j2.ToString;
            sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf(vname2)] + ',';
```

```
        end;

        for j2 := 1 to Config.YearsOfAnalysis do
        begin
          if UseSavedBins then
            sLineOut := sLineOut +
PWSSampleBins.GetDataValue('pp_lsl_replaced_vol_pct' + j2.ToString, SLSavedData) +
','
          else
            sLineOut := sLineOut + LSLVolPctRates[j2].ToString + ',';
        end;

        for j2 := 1 to Config.YearsOfAnalysis do
        begin
          if UseSavedBins then
            sLineOut := sLineOut + PWSSampleBins.GetDataValue('BinChg_' +
j2.ToString, SLSavedData) + ','
          else
            sLineOut := sLineOut + arrBinChg[j2].ToString + ',';
        end;

        // bBin has values 1 - 5
        sLineOut := sLineOut + bBin.ToString + ',';
        sLineOut := sLineOut + LCRx_Bin.ToString + ',';
        sLineOut := sLineOut + LCRx_P90.ToString + ',';
        sLineOut := sLineOut + LCRx_Bin_15.ToString + ',';
        sLineOut := sLineOut + LCRx_Bin_12.ToString + ',';
        sLineOut := sLineOut + LCRx_Bin_10.ToString + ',';
        sLineOut := sLineOut + LCRx_Bin_5.ToString + ',';
        sLineOut := sLineOut + BaseBin.ToString + ',';
        sLineOut := sLineOut + bALENotAchieved.ToString + ',';

    sLineOut := Copy(sLineOut, 1, Length(sLineOut)-1);
    Writer.WriteLine(sLineOut);
  end;
begin
  if UseSavedBins then
  begin
    PWSSampleBins.ReadBinFile(false, LSLLevel, SmallProxyLabel);
  end;

  inFileName := ExtractFilePath(SDWISFile) + 'LCRBaseline_' + SampleName + '.csv';
  Reader := TStreamReader.Create(inFileName);

  outFileName := ExtractFilePath(SDWISFile) + OptionName + SampleName + '.csv';
  OptionSampleFilename := outFileName;
  Writer := TStreamWriter.Create(outFileName);

  slInData := TStringList.Create;
  slInData.Delimiter := ',';
```

```
slInData.StrictDelimiter := true;
slInCols := TStringList.Create;
slInCols.Delimiter := ',';
slInCols.StrictDelimiter := true;

// write column headings for outFile
     sLineOut := '';
     sLineOut := sLineOut + 'PWSID,';
     sLineOut := sLineOut + 'SystemType,';
     sLineOut := sLineOut + 'Population,';
     sLineOut := sLineOut + 'Connections,';
     sLineOut := sLineOut + 'SourceWater,';
     sLineOut := sLineOut + 'CCT,';
     sLineOut := sLineOut + 'PopCat,';
     sLineOut := sLineOut + 'State,';
     sLineOut := sLineOut + 'Owner,';
     sLineOut := sLineOut + 'First_ale,';
     sLineOut := sLineOut + 'Weight,';
     sLineOut := sLineOut + 'NumberEPs,';
     sLineOut := sLineOut + 'LSL,';
     sLineOut := sLineOut + 'NumberLSLs,';
     sLineOut := sLineOut + 'CCTPO4,';
     sLineOut := sLineOut + 'CCTPH,';
     sLineOut := sLineOut + 'CCTBoth,';
     sLineOut := sLineOut + 'BaselinePO4Dose,';
     sLineOut := sLineOut + 'Baselineph_wPh,';
     sLineOut := sLineOut + 'Baselineph_woPh,';
     sLineOut := sLineOut + 'Baselineph_woCCT,';
     sLineOut := sLineOut + 'Baselineph_wPO4Ph,';

     sLineOut := sLineOut + 'Bin,';
     sLineOut := sLineOut + 'Small_Correct,';
     sLineOut := sLineOut + 'Num_Proxies,';

     for cv in ScenCostSteps.CostVars.Values do
     begin
       vname := cv.fID;
       sLineOut := sLineOut + vname + ',';
       if cv.fImAProb then
       begin
         sLineOut := sLineOut + vname + '_r,';
       end;
     end;

     for j := 4 to Config.YearsOfAnalysis do
     begin
       vname := 'pp_lsl_replacement_' + j.ToString;
       sLineOut := sLineOut + vname + ',';
     end;
```

```
      for j := 1 to Config.YearsOfAnalysis do
      begin
        vname := 'p_source_chng_' + j.ToString;
        sLineOut := sLineOut + vname + ',';
      end;

      for j := 1 to Config.YearsOfAnalysis do
      begin
        vname := 'p_source_sig_' + j.ToString;
        sLineOut := sLineOut + vname + ',';
      end;

      for j := 1 to Config.YearsOfAnalysis do
      begin
        vname := 'p_treat_change_' + j.ToString;
        sLineOut := sLineOut + vname + ',';
      end;

      for j := 1 to Config.YearsOfAnalysis do
      begin
        vname := 'pp_lsl_replaced_vol_pct_' + j.ToString;
        sLineOut := sLineOut + vname + ',';
      end;

      for j := 1 to Config.YearsOfAnalysis do
      begin
        vname := 'BinChg_' + j.ToString;
        sLineOut := sLineOut + vname + ',';
      end;

      sLineOut := sLineOut + 'bBin,';
      sLineOut := sLineOut + 'LCRx_Bin,';
      sLineOut := sLineOut + 'LCRx_P90,';
      sLineOut := sLineOut + 'LCRx_Bin_15,';
      sLineOut := sLineOut + 'LCRx_Bin_12,';
      sLineOut := sLineOut + 'LCRx_Bin_10,';
      sLineOut := sLineOut + 'LCRx_Bin_5,';
      sLineOut := sLineOut + 'BaseBin,';
      sLineOut := sLineOut + 'bALENotAchieved,';

      sLineOut := Copy(sLineOut, 1, Length(sLineOut)-1);
      Writer.WriteLine(sLineOut);

// read first row of inFile to get column headings
sLineIn := Reader.ReadLine;
slInCols.CommaText := sLineIn;

hpwsid := '';
r := 1;
while not Reader.EndOfStream do
```

```
  begin
    sLineIn := Reader.ReadLine;

    slInData.CommaText := sLineIn;

    pwsid := slInData.Strings[slInCols.IndexOf('PWSID')];
    sProxy := RightStr(pwsid,1);
    if (sProxy = 'a') or (sProxy = 'b') or (sProxy = 'c') then
      pwsid :=  Copy(pwsid, 1, Length(pwsid)-1);

    if hpwsid = pwsid then continue;
    hpwsid := pwsid;

    if slInData.Strings[slInCols.IndexOf('SourceWater')] = 'Groundwater' then
      SrcWater := 1
    else
      SrcWater := 2;

    SystemType := slInData.Strings[slInCols.IndexOf('SystemType')].ToInteger;
    SizeCat := slInData.Strings[slInCols.IndexOf('PopCat')].ToInteger;

    population := slInData.Strings[slInCols.IndexOf('Population')].ToInteger;

    LSL := slInData.Strings[slInCols.IndexOf('LSL')].ToInteger;
    if LSL > 0 then
      pws_lsl := 1
    else
      pws_lsl := 0;

    CCT := slInData.Strings[slInCols.IndexOf('CCT')].ToInteger;

    bBin := slInData.Strings[slInCols.IndexOf('bBin')].ToInteger;
    baseBin := bBin;
    ScenCostSteps.DetermineSystemPValues(SizeCat, SrcWater, pws_lsl, CCT,
SystemType, False, 'OH2504412', False, True);

    bBin := AssignOptionBin2(pws_lsl, ScenCostSteps.PWS_p_values, bBin, OptionName);

    if OptionName = 'LCRR' then
    begin
      if bBin = 1 then Bin := 1
      else if (bBin = 2) or (bBin = 3) then Bin := 2
      else Bin := 3;

      if Bin = 3 then LCRx_P90 := 3
      else if Bin = 2 then LCRx_P90 := 12
      else LCRx_P90 := 20;
    end
    else if OptionName = 'LCRI' then
    begin
```

```
      if bBin = 1 then LCRx_BIN_15 := 1 else LCRx_BIN_15 := 3;
      if (bBin >= 1) and (bBin <= 2) then LCRx_BIN_12 := 1 else LCRx_BIN_12 := 3;
      if (bBin >= 1) and (bBin <= 3) then LCRx_BIN_10 := 1 else LCRx_BIN_10 := 3;
      if (bBin >= 1) and (bBin <= 4) then LCRx_BIN_5 := 1 else LCRx_BIN_5 := 3;

      if bBin = 3 then LCRx_P90 := 3;
      if bBin = 1 then LCRx_P90 := 20;
    end;

    ScenCostSteps.ResolveDatabaseVariables(SizeCat, SrcWater, pws_lsl, CCT,
SystemType,
                                           PWS90PctBp1, PWS90PctBp2, slInCols,
slInData);

    if not UseSavedBins then
    begin
      rr := iiRandom(rrScen);
      if rr < 0.15 then bALENotAchieved := 1
      else bALENotAchieved := 0;

      SetLength(LSLReplacementRates, Config.YearsOfAnalysis + 1);
      for j := Low(LSLReplacementRates) to High(LSLReplacementRates) do
LSLReplacementRates[j] := 0;
      ScenCostSteps.CostVars.DrawLSLReplacementRates(SizeCat, SrcWater,
                                                     pws_lsl, CCT, SystemType,
Config.YearsOfAnalysis, OptionName,
                                                     LSLReplacementRates);

      SetLength(LSLVolPctRates, Config.YearsOfAnalysis + 1);
      for j := Low(LSLVolPctRates) to High(LSLVolPctRates) do LSLVolPctRates[j] :=
0;
      ScenCostSteps.CostVars.DrawPp_LSL_Replaced_Vol_Pct(SizeCat, SrcWater,
                                                         pws_lsl, CCT, SystemType,
Config.YearsOfAnalysis, '', LSLVolPctRates);

      SetLength(arrBinChg, Config.YearsOfAnalysis + 1);

      for j := 1 to Config.YearsOfAnalysis do
      begin
        nn1 := 'p_source_chng_' + j.ToString;
        nn2 := 'p_treat_change_' + j.ToString;
        arrBinChg[j] := 0;
        if (slInData.Strings[slInCols.IndexOf(nn1)] = '1') or
(slInData.Strings[slInCols.IndexOf(nn2)] = '1') then
        begin
          arrBinChg[j] := ScenCostSteps.AssignNewBinOption(OptionName);
        end;
      end;
    end;
```

```
{
  for small systems Pop <= 3300
  LSL   CCT     Small_Correct   Num_Proxies
  0       0,1     2,3                   2
  1       0,1     1,2,3               3

  for other systems Num_Proxies = 0
}
     if ProxyRecords then begin
       if (SystemType = 1) and (population > SmallProxyPop) then
         WriteLine(Bin,0,0,'')
       else begin
         if (pws_lsl = 0) then begin
           WriteLine(Bin,2,2,'b');
           WriteLine(Bin,3,2,'c');
         end
         else if (pws_lsl = 1)then begin
           WriteLine(Bin,1,3,'a');
           WriteLine(Bin,2,3,'b');
           WriteLine(Bin,3,3,'c');
         end
       end;
     end else
       WriteLine(Bin,0,0,'');

    inc(r);
  end;

  Reader.Free;
  Writer.Free;

  slInData.Free;
  slInCols.Free;
end;

end.
```