```
unit PWSReplicator;

interface

uses Classes, SysUtils, Math, LCRConfig, CostingSteps, LCRGlobals, VLSSystemData,
LCRCostVars,
     PWS_Sample_Bins;

type

TPWSReplicator = class(TObject)
public
  SDWISFile: string;
  OptionName: string;
  SampleName: string;
  MakeProfileSample : boolean;
  SampRate : double;
  MinPerCategory: integer;
  NoReplication: boolean;
  Config: TLCRConfig;
  BaseCostVars: TCostVars;
  BaseCostSteps: 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..1, 0..1, 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;

  { 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 WriteBaseline3;
  procedure WriteOption;

  constructor Create;
  destructor Destroy; override;
private
  VLSSystemData: TVLSSystemData;
  PWSSampleBins: TPWSSampleBins;

  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): integer; overload;
  function AssignOptionBin(LSL: integer; p_values: TPWS_p_valuesRec; baseBin:
integer; Option: string): integer; overload;
end;

implementation

uses VCL.FlexCel.Core, FlexCel.XLSAdapter;

{ TPWSReplicator }

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

```
  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): integer;
var
  r: double;
  i, iBin: integer;
  tmp: double;
  arrPBin_loc: array[0..2] of double;
begin
  arrPBin_loc[0] := p_values.ppBin1;
  arrPBin_loc[1] := p_values.ppBin2;
  arrPBin_loc[2] := p_values.ppBin3;

  r := iiRandom(true);

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

  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(false);
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;
  pAdjBin1, pAdjBin2: double;
  pBin1, pBin2, pBin3: double;
  arrPBin_loc: array[1..3] of double;
  arrPAdjBin_loc: array[1..3] of double;
  pTestAdjBin1: double;
  pBin2ToBin1, pBin3ToBin1, pBin2Remain, pBin3Remain: double;
begin
  arrPBin_loc[1] := p_values.ppBin1;
  arrPBin_loc[2] := p_values.ppBin2;
  arrPBin_loc[3] := p_values.ppBin3;

  r := iiRandom(false);
  iBin := 0;

  arrPAdjBin_loc[1] := p_values.ppAdjBin1;
  arrPAdjBin_loc[2] := p_values.ppAdjBin2;
  arrPAdjBin_loc[3] := p_values.ppAdjBin3;
  pBin1 := arrPBin_loc[1];
  pBin2 := arrPBin_loc[2];
  pBin3 := arrPBin_loc[3];
  pAdjBin1 := arrPAdjBin_loc[1];
  pAdjBin2 := arrPAdjBin_loc[2];

  if LSL = 0 then
    iBin := baseBin
  else begin
    if baseBin = 1 then
      iBin := 1
    else
    if baseBin = 2 then begin
      p := pAdjBin1 - pBin1;
      if p >= pBin2 then
        p := 1
      else
        p := (pAdjBin1 - pBin1)/pBin2;
      if r<p then iBin:=1 else iBin:=2;
    end else begin
      pBin3ToBin1 := 0;
```

```
      pBin2ToBin1 := 0;

      p := pAdjBin1 - pBin1;
      if p >= pBin2 then
      begin
        p := (pAdjBin1 - pBin1 - pBin2) / pBin3;
        if r<p then iBin:=1 else iBin:=3;
        //csl('basebin=3 to 1 ' + p.ToString + ' ' + r.ToString + ' ' +
iBin.ToString);
      end;

      if iBin <> 1 then
      begin
        pBin2Remain := max(pBin2 - ((pAdjBin1 - pBin1)/Pbin2), 0);
        p := (pAdjBin2 - pBin2Remain)/Pbin3;
        r := iiRandom(false);
        if r<p then iBin:=2 else iBin:=3;
      end;
    end;
  end;

  Result := iBin;
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

  // 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;

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

destructor TPWSReplicator.Destroy;
begin
  VLSSystemData.Free;
  PWSSampleBins.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.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;
begin
  if SDWISFile = '' then
    exit;

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

  for r := 2 to Xls.RowCount do begin
     SizeCatSort := xls.GetCellValue(r, 24).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 = 'Ground water' then
        GwSw := 'GW' else
      if GwSw = 'Surface water' then
        GwSw := 'SW';

      if GwSw = '-' then
        continue;

      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;

      num_pop := xls.GetCellValue(r, 7).AsVariant;
      if num_pop < 25 then num_pop := 25;

      connections := xls.GetCellValue(r, 10).AsVariant;

      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
        CategoryRep[st,i,j] := Ceil(MinPerCategory / CategoryCount[st,i,j]);
        if NoReplication then
          CategoryWeight[st,i,j] := 1
        else
          CategoryWeight[st,i,j] := 1 / CategoryRep[st,i,j];

        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.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;

  Outfile: string;

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

  Writer: TStreamWriter;
  sLine: string;

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

  Bin: integer;
  BinChg: integer;

  SLSavedData: TStringList;
begin
  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;

  for r := 1 to XlsIn.RowCount do begin
     if r > 1 then
     begin
       if MakeProfileSample then begin
         if Random>SampRate then continue;
       end;
```

```
// exclude systems serving > 1,000,000
// begin including them 3/19/18
SizeCatSort := XlsIn.GetCellValue(r, 24).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 = 'Ground water' then
  GwSw := 'GW' else
if GwSw = 'Surface water' then
  GwSw := 'SW';

if GwSw = '-' then
  continue;

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, 23) = 'FALSE' then
  CCT := 0
else
  CCT := 1;

if xlsIn.GetStringFromCell(r, 34) = 'FALSE' then
  first_ale := 0
else
  first_ale := 1;

population := xlsIn.GetCellValue(r, 7).AsVariant;
```

```
      if population < 25 then population := 25;

      connections := xlsIn.GetCellValue(r, 10).AsVariant;

      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;

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

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

      sLine := Copy(sLine, 1, Length(sLine)-1);
      Writer.WriteLine(sLine);
    end
    else
    begin
      for iRep := 1 to getRepFactor(SystemType, SizeCatSort, SrcWater) do
      begin

        BaseCostSteps.DetermineSystemPValues(SizeCatSort, SrcWater, 1, CCT,
SystemType, True, 'OH2504412');
        p_lsl := BaseCostSteps.PWS_p_values.p_lsl;

        Bin := AssignBaselineBin(BaseCostSteps.PWS_p_values);

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

        num_lsl := min(population, num_connect) *
BaseCostSteps.PWS_p_values.perc_lsl * p_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);
          sLine := '';
          sLine := sLine + xlsIn.GetStringFromCell(r, 1) + '_' + iRep.ToString + ',';
          sLine := sLine + SystemType.ToString + ',';
          sLine := sLine + population.ToString + ',';
          sLine := sLine + num_connect.ToString + ',';
          sLine := sLine + xlsIn.GetStringFromCell(r, 12) + ',';
          sLine := sLine + CCT.ToString + ',';
          sLine := sLine + SizeCatSort.ToString + ',';
          sLine := sLine + xlsIn.GetStringFromCell(r, 36) + ',';
          sLine := sLine + xlsIn.GetStringFromCell(r, 37) + ',';
          sLine := sLine + first_ale.ToString + ',';
          sLine := sLine + getWeight(SystemType, SizeCatSort, SrcWater).ToString +
',';


          if PWSId = 'OH2504412' then
          begin
            numeps := 30;
            sLine := sLine + '30' + ',';
          end
          else if SizeCatSort < 9 then
          begin
            if SystemType = 1 then
              numeps := getEP(SizeCatSort-1, SrcWater-1)
            else
              numeps := 1;
            sLine := sLine + numeps.ToString + ',';
          end
          else
          begin
            VLSSystemData.GetSystemData(PWSId);
            numeps := VLSSystemData.Num_EP;
            sLine := sLine + numeps.ToString + ',';
          end;

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

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

          sLine := sLine + num_lsl.ToString + ',';
          sLine := sLine + pbasepo4.ToString + ',';
```

```
          sLine := sLine + pbaseph.ToString + ',';
          sLine := sLine + pbasephpo4.ToString + ',';
          sLine := sLine + baselinepo4dose.ToString + ',';
          sLine := sLine + baselineph_wph.ToString + ',';
          sLine := sLine + baselineph_woph.ToString + ',';
          sLine := sLine + baselineph_wocct.ToString + ',';
          sLine := sLine + baselineph_wpo4ph.ToString + ',';

          if UseSavedBins then
            sLine := sLine + PWSSampleBins.GetDataValue('Bin',SLSavedData) + ','
          else
            sLine := sLine + Bin.ToString + ',';

          sLine := sLine + '0,';
          sLine := sLine + '0,';

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

          SetLength(RepRates, Config.YearsOfAnalysis + 1);
          SetLength(SrcRepRates, Config.YearsOfAnalysis + 1);
          SetLength(SigRepRates, Config.YearsOfAnalysis + 1);
          SetLength(TrtRepRates, 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
              sLine := sLine + PWSSampleBins.GetDataValue('pp_lsl_replacement_' +
j.ToString, SLSavedData) + ',';

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

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

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

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

          for j := 1 to Config.YearsOfAnalysis do
            sLine := sLine + PWSSampleBins.GetDataValue('BinChgNoLSL_' +
j.ToString, SLSavedData) + ',';
        end
        else
        begin
          BaseCostSteps.CostVars.DrawLSLReplacementRates(SizeCatSort, SrcWater,
                                                         p_lsl, CCT, SystemType,
Config.YearsOfAnalysis, 'Baseline',
                                                         RepRates);

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

          for j := Low(SrcRepRates) to High(SrcRepRates) do SrcRepRates[j] := 0;
          BaseCostSteps.CostVars.DrawP_Source_Chng(SizeCatSort, SrcWater,
                                                   p_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,
                                                   p_lsl, CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', SigRepRates);

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

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

          for j := Low(TrtRepRates) to High(TrtRepRates) do TrtRepRates[j] := 0;
          BaseCostSteps.CostVars.DrawP_Treat_Change(SizeCatSort, SrcWater,
                                                    p_lsl, CCT, SystemType,
Config.YearsOfAnalysis, numeps, '', TrtRepRates);

          for j := 1 to Config.YearsOfAnalysis do
            sLine := sLine + 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(CCT, 1);
            end;
            sLine := sLine + BinChg.ToString + ',';
          end;

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

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

  RepRates: TDoubleArray;
```

```
  LSLReplacementRates: TDoubleArray;
  LSLVolPctRates: TDoubleArray;
  BinChgLSL: TDoubleArray;
  BinChgNoLSL: TDoubleArray;

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

  procedure WriteLine(Bin, SmallCorrect, NumProxies: integer; suffix: string);
  var
    cv2: TCostVar;
    j2: integer;
    vname2: string;
    CCT, LSL: integer;
    SystemType: integer;
    xBin: string;
  begin
    SystemType := slInData.Strings[slInCols.IndexOf('SystemType')].ToInteger;
    CCT := slInData.Strings[slInCols.IndexOf('CCT')].ToInteger;
    LSL := slInData.Strings[slInCols.IndexOf('LSL')].ToInteger;

    sLineOut := '';
    sLineOut := sLineOut + slInData.Strings[slInCols.IndexOf('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('BinChgLSL_' +
j2.ToString, SLSavedData) + ','
            else
              sLineOut := sLineOut + BinChgLSL[j2].ToString + ',';
          end;

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

      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) + 'Baseline_' + 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 := 'BinChgLSL_' + j.ToString;
  sLineOut := sLineOut + vname + ',';
end;

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

```
      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;

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

    slInData.CommaText := sLineIn;

    if slInData.Strings[slInCols.IndexOf('SourceWater')] = 'Ground water' 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;
    CCT := slInData.Strings[slInCols.IndexOf('CCT')].ToInteger;

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

    ScenCostSteps.DetermineSystemPValues(SizeCat, SrcWater, 1, CCT, SystemType,
True, 'OH2504412');

    Bin := AssignOptionBin(LSL, ScenCostSteps.PWS_p_values, baseBin, OptionName);

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

    if not UseSavedBins then
    begin
      SetLength(LSLReplacementRates, Config.YearsOfAnalysis + 1);
      for j := Low(LSLReplacementRates) to High(LSLReplacementRates) do
LSLReplacementRates[j] := 0;
      ScenCostSteps.CostVars.DrawLSLReplacementRates(SizeCat, SrcWater,
                                                     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,
                                                         LSL, CCT, SystemType,
Config.YearsOfAnalysis, '', LSLVolPctRates);

      SetLength(BinChgLSL, Config.YearsOfAnalysis + 1);
      SetLength(BinChgNoLSL, Config.YearsOfAnalysis + 1);

      for j := 1 to Config.YearsOfAnalysis do
      begin
        nn1 := 'p_source_chng_' + j.ToString;
        nn2 := 'p_treat_change_' + j.ToString;
        BinChgLSL[j] := 0;
        BinChgNoLSL[j] := 0;
        if (slInData.Strings[slInCols.IndexOf(nn1)] = '1') or
(slInData.Strings[slInCols.IndexOf(nn2)] = '1') then
        begin
          BinChgLSL[j] := ScenCostSteps.AssignNewBinOption(CCT, 1, OptionName);
          BinChgNoLSL[j] := ScenCostSteps.AssignNewBinOption(CCT, 0, 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 (LSL = 0) then begin
           WriteLine(Bin,2,2,'b');
           WriteLine(Bin,3,2,'c');
         end
         else if (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.
```