```
unit LCRPWSRecords;

interface

uses LCRGlobals, DB, Classes, LCRConfig,Math,
     LCRPopulation, Generics.Collections;

type

  TPWSRecordObj = class
  public
     PWSId : string[200];
     State : string[2];
     SystemSize : TSystemSize;
     SystemType : TSystemType;
     Ownership : TOwnership;
     SourceWater : TSourceWater;
     Region,
     EPARegion : Integer;
     CurCost : integer;
     InMap,RecNo : Integer;
     RunIt : boolean;

     Population : integer;
     SamplingWeight : double;
     InflatedPopulation : TSingleArray;

     SubPops            : TYearlySubPopArray;
     CategoryMembership : string;

     StateIdx: integer;
     AvgRevenue,CostCapital,PWSAnnualRevenue : double;
     Data1,Data2 : double;

     CCT: integer;
     Connections: integer;
     First_ale: integer;

     NumberEPs: integer;
     LSL: integer;
     NumberLSLs: double;
     CCTPO4, CCTPH, CCTBoth, BaselinePO4Dose: integer;
     Baselineph_wPh, Baselineph_woPh, Baselineph_woCCT, Baselineph_wPO4Ph: integer;
     Small_Correct: integer;
     Num_Proxies: integer;
     Bin: integer;

     Cost : double;
```

```
     fVars : TDictionary<string,double>;
     fVarNames : TstringList;

     constructor Create;
     destructor Destroy();override;

     procedure GetFlows(const EP : integer; Yr : integer; config: TLCRConfig; var
AFlow,DFlow : double);
  end;


  TPWSRecords = class(TObject)
  private
     fConfig : TLCRConfig;
     fBasePWS,fScenPWS : TPWSRecordObj;
     fQBasePWS,fQScenPWS : array[0..10] of TPWSRecordObj;

     fBaseFileFS, fScenFileFS : TBufferedFileStream;
     fBaseFile, fScenFile : TStreamReader;

     fBCurNumProxies,fSCurNumProxies,
     fBRecCnt,fSRecCnt,
     fBCurNumProxiesCnt,fSCurNumProxiesCnt : integer;

     fTS : TStringList;

     function ConvertOwnerType(S: string): string;
  public
     UserRandSeedS, UserRandSeedB : integer;

     constructor Create(config: TLCRConfig);
     destructor Destroy();override;

     procedure OpenFromCSVPair(BaseCSV,ScenCSV : string; MaxRecs: integer);
     function Next : boolean;

     property CurBasePWS : TPWSRecordObj read fBasePWS;
     property CurScenPWS : TPWSRecordObj read fScenPWS;
  end;

implementation

uses SysUtils, VCL.FlexCel.Core, FlexCel.XlsAdapter, Dialogs;

{ TPWSRecords }
```

```
function TPWSRecords.ConvertOwnerType(S: string): string;
begin
  if S = 'Private' then
    Result := 'Private'
  else
    Result := 'Public';
end;

constructor TPWSRecords.Create(config: TLCRConfig);
begin
  fConfig := config;
  fBasePWS:=nil;
  fScenPWS:=nil;
  fBaseFileFS:=nil;
  fScenFileFS:=nil;
  fBaseFile:=nil;
  fScenFile:=nil;
  UserRandSeedS:=0;
  UserRandSeedB:=0;
end;

destructor TPWSRecords.Destroy;
var i : integer;
begin
  for i:=0 to high(fQBasePWS) do
    if Assigned(fQBasePWS[i]) then fQBasePWS[i].Free;
  for i:=0 to high(fQScenPWS) do
    if Assigned(fQScenPWS[i]) then fQScenPWS[i].Free;

  if Assigned(fBaseFile) then fBaseFile.Free;
  if Assigned(fScenFile) then fScenFile.Free;
  if Assigned(fBaseFileFS) then fBaseFileFS.Free;
  if Assigned(fScenFileFS) then fScenFileFS.Free;
  if Assigned(fTS) then fTS.Free;
  inherited;
end;

procedure TPWSRecords.OpenFromCSVPair(BaseCSV, ScenCSV: string; MaxRecs: integer);
var a,i : integer;
    s: string;
begin
  fBCurNumProxies:=0;
  fSCurNumProxies:=0;
  fBCurNumProxiesCnt:=0;
  fSCurNumProxiesCnt:=0;
  fBRecCnt:=0;
  fSRecCnt:=0;
```

```
  fTS:=TStringList.Create;
  fTS.Delimiter:=',';
  fTS.StrictDelimiter:=True;

  if BaseCSV<>'' then begin
    fBaseFileFS := TBufferedFileStream.Create(BaseCSV,fmOPenRead + fmShareDenyNone);
    fBaseFile := TStreamReader.Create(fBaseFileFS, TEncoding.ASCII, False, 4096);
    s:=fBaseFile.ReadLine;
    fTS.CommaText:=s;

    for i:=0 to high(fQBasePWS) do begin
      fQBasePWS[i]:=TPWSRecordObj.Create;
      for a:=19 to fTS.Count-1 do begin
        fQBasePWS[i].fVars.AddOrSetValue(fTS[a],0);
      end;
      for a:=0 to fTS.Count-1 do begin
        fQBasePWS[i].fVarNames.Add(fTS[a]);
      end;
    end;

  end;
  if ScenCSV<>'' then begin
    fScenFileFS := TBufferedFileStream.Create(ScenCSV,fmOPenRead + fmShareDenyNone);
    fScenFile := TStreamReader.Create(fScenFileFS, TEncoding.ASCII, False, 4096);
    s:=fScenFile.ReadLine;
    fTS.CommaText:=s;
    for i:=0 to high(fQScenPWS) do begin
      fQScenPWS[i]:=TPWSRecordObj.Create;
      for a:=19 to fTS.Count-1 do begin
        fQScenPWS[i].fVars.AddOrSetValue(fTS[a],0);
      end;
      for a:=0 to fTS.Count-1 do begin
        fQScenPWS[i].fVarNames.Add(fTS[a]);
      end;
    end;
  end;

end;

function TPWSRecords.Next: boolean;
var s : string;
    tmp : integer;
    tCost : double;
    cc,vv : integer;
    PriScenPWSID : string;
    Synced : boolean;

  function GetStateIdx(State: string): integer;
```

```
var
  i: integer;
begin
  Result := -1;
  for i := 0 to length(fConfig.StateDataArray) - 1 do
    if string(fConfig.StateDataArray[i].StateCode) = State then
    begin
      Result := i;
      break;
    end;
end;

procedure SetPWS(var P : TPWSRecordObj);
var GwSw,PWSType,SystemType : string;
    population : integer;
    Owner,tn: string;
    v,c : integer;
    tv : double;
begin
  SystemType:='';
  if fConfig.SystemType = sysCWS then
    SystemType := 'CWS'
  else
  if fConfig.SystemType = sysNTNC then
    SystemType := 'NTNCWS';

  // exclude invalid GW or SW Code
  GwSw := fTS[4];
  if GwSw = 'Ground water' then
   GwSw := 'GW' else
  if GwSw = 'Surface water' then
   GwSw := 'SW';

  if GwSw = '-' then begin
    P.RunIt:=False;
    exit;
  end;

  Owner := ConvertOwnerType(fTS[8]);
  population := strtoint(fTS[2]);
  if population < 25 then population := 25;

  P.Data1:=0;
  P.Data2:=0;
  P.PWSId  := fTS[0];
  P.State:=fTS[7];
  P.Population     := population;
  P.SamplingWeight := strtofloat(fTS[10]);
```

```
P.SystemSize     := ConvertToTSystemSize(P.Population);

if fTS[1] = '1' then
  P.SystemType := sysCWS
else
  P.SystemType := sysNTNC;

P.Ownership      := StrToTOwnership(Owner);
P.SourceWater    := StrToTSourceWater(GwSw);
P.Connections    := strtoint(fTS[3]);

P.CCT:=strtoint(fTS[5]);
P.First_ale:=strtoint(fTS[9]);
P.NumberEPs :=  strtoint(fTS[11]);
P.LSL := strtoint(fTS[12]);
P.NumberLSLs := strtofloat(fTS[13]);

P.CCTPO4 := strtoint(fTS[14]);
P.CCTPH := strtoint(fTS[15]);
p.CCTBoth := strtoint(fTS[16]);
P.BaselinePO4Dose := strtoint(fTS[17]);
P.BaselinePH_wPh := strtoint(fTS[18]);
P.BaselinePH_woPh := strtoint(fTS[19]);
P.BaselinePH_woCCT := strtoint(fTS[20]);
P.BaselinePH_wPO4Ph := strtoint(fTS[21]);
P.Bin := strtoint(fTS[22]);
P.Small_Correct := strtoint(fTS[23]);
P.Num_Proxies := strtoint(fTS[24]);
P.Cost:=0;
P.AvgRevenue         := fConfig.GetAvgRevenue(P.Ownership,P.Population);

//!!!!!!!! HERE  -  Assumes the selection from Config is not changed
c:=Integer(P.SourceWater) * 9 + Integer(P.SystemSize)+1;
c:=c+4;

if (P.Ownership = oPrivate) and (P.SystemType=sysNTNC) then c:=c+18
else if (P.Ownership = oPublic) and (P.SystemType=sysCWS) then c:=c+36
else if (P.Ownership = oPublic) and (P.SystemType=sysNTNC) then c:=c+54;

if P.SystemType=sysNTNC then
begin
  if P.Ownership = oPublic then
    P.CategoryMembership := '0,2,4,'+inttostr(c)
  else
    P.CategoryMembership := '0,2,3,'+inttostr(c)
end
else
if P.SystemType=sysCWS then
```

```
    begin
      if P.Ownership = oPublic then
        P.CategoryMembership := '0,1,4,'+inttostr(c)
      else
        P.CategoryMembership := '0,1,3,'+inttostr(c)
    end
    else
      P.CategoryMembership := '0,'+inttostr(c); //TODO this is really an error isn't
it?

    P.CostCapital        := fConfig.GetCostOfCapital(P.Ownership,P.Population);
    P.RunIt:=True;

    P.StateIdx := GetStateIdx(String(P.State));

   // Count the number of PWS in each state
    if P.StateIdx > 0 then
      fConfig.StateDataArray[P.StateIdx].PWSCount :=
fConfig.StateDataArray[P.StateIdx].PWSCount + 1;

    for v:=25 to fTS.Count-1 do begin
      tn:=P.fVarNames[v];
      try
        tv:=strtofloat(fTS[v]);
      except
        on e: Exception do
        begin
          ShowMessage(e.Message + ' ' + v.ToString + ' ' + tn);
        end;
      end;
      if copy(tn,1,5)='post_' then begin
        if tv=1 then tv:=12 else
        if tv=2 then tv:=5;
      end;
      P.fVars.AddOrSetValue(tn,tv);
    end;
  end;

begin
  if Assigned(fScenFile) then begin
    if (fSRecCnt>0) and (fScenPWS.Num_Proxies>0) and
(fScenPWS.Num_Proxies=fSCurNumProxiesCnt) then begin
      tCost:=9e99;
      for cc:=0 to fSCurNumProxiesCnt-1 do begin
        if fQScenPWS[cc].Cost<tCost then begin
          vv:=cc;
          tCost:=fQScenPWS[cc].Cost;
        end;
```

```
      end;
      fScenPWS:=fQScenPWS[vv];
      fScenPWS.Num_Proxies:=0;
      fSCurNumProxiesCnt:=0;
    end else begin
      if fScenFile.EndOfStream then begin
        Result:=False;
        exit;
      end;
      s:=fScenFile.ReadLine;
      inc(fSRecCnt);
      fTS.CommaText:=s;

      //!!!!Note num_proxies being forced here...
      tmp:=strtoint(fTS[24]);
      if tmp>0 then begin
        tmp:=fSCurNumProxiesCnt;
        inc(fSCurNumProxiesCnt);
      end;
      SetPWS(fQScenPWS[tmp]);
      fScenPWS:=fQScenPWS[tmp];
    end;
    UserRandSeedS:=SetRandSeed(fScenPWS.PWSId);
    PriScenPWSID:=RemoveProxyLetter(fScenPWS.PWSId);
  end;

  //This is another adhoc adjustment to deal with the new Proxy records in options.
  //TODO: Probably best to deal with this in the sample creation.
  Synced:=False;
  if Assigned(fBasePWS) then
    if Assigned(fScenFile) then
      if fBasePWS.PWSId=PriScenPWSID then Synced:=True;

  if Assigned(fBaseFile) and (not Synced) then begin
    if (fBRecCnt>0) and (fBasePWS.Num_Proxies>0) and
(fBasePWS.Num_Proxies=fBCurNumProxiesCnt) then begin
      tCost:=9e99;
      for cc:=0 to fBCurNumProxiesCnt-1 do begin
        if fQBasePWS[cc].Cost<tCost then begin
          vv:=cc;
          tCost:=fQBasePWS[cc].Cost;
        end;

      end;
      fBasePWS:=fQBasePWS[vv];
      fBasePWS.Num_Proxies:=0;
      fBCurNumProxiesCnt:=0;
    end else begin
```

```
      if fBaseFile.EndOfStream then begin
        Result:=False;
        exit;
      end;
      s:=fBaseFile.ReadLine;
      inc(fBRecCnt);
      fTS.CommaText:=s;
      //!!!!Note num_proxies location being forced here...
      tmp:=strtoint(fTS[24]);
      if tmp>0 then begin
        tmp:=fBCurNumProxiesCnt;
        inc(fBCurNumProxiesCnt);
      end;
      SetPWS(fQBasePWS[tmp]);
      fBasePWS:=fQBasePWS[tmp];
    end;
    UserRandSeedB:=SetRandSeed(fBasePWS.PWSId);
  end;

  Result:=True;
  if Assigned(fScenFile) and Assigned(fBaseFIle) then begin
    if PriScenPWSID<>fBasePWS.PWSId then
      raise exception.Create('Mismatch');
  end;

end;

{ TPWSRecordObj }

constructor TPWSRecordObj.Create;
begin
  fVars:=TDictionary<string,double>.create;
  fVarNames := TstringList.Create;
end;


destructor TPWSRecordObj.Destroy;
begin
  fVars.Free;
  fVarNames.Free;
  inherited;
end;

procedure TPWSRecordObj.GetFlows(const EP: integer; Yr : integer; config:
TLCRConfig; var AFlow, DFlow : double);
var fAvgFlowA,fAvgFlowB,fDesignFlowA,fDesignFlowB,dtmp : double;
    fPopulation : double;
begin
```

```
  fAvgFlowA:=Config.FlowVars[Integer(Ownership)+1,Integer(SourceWater)+1,1].V;
  fAvgFlowB:=Config.FlowVars[Integer(Ownership)+1,Integer(SourceWater)+1,2].V;
  fDesignFlowA:=Config.DFlowVars[Integer(Ownership)+1,Integer(SourceWater)+1,1].V;
  fDesignFlowB:=Config.DFlowVars[Integer(Ownership)+1,Integer(SourceWater)+1,2].V;

  //!!!!!!!TODO this has to be set by year....
  fPopulation := Population;

  AFlow := ( fAvgFlowA * Power(fPopulation,fAvgFlowB) * 0.001 ) / EP;
  dtmp := ( fDesignFlowA * Power(fPopulation,fDesignFlowB) * 0.001 ) / EP;
  DFlow := Max(2*AFlow,dtmp);

  PWSAnnualRevenue:=AvgRevenue * (AFlow * 365000);
end;


end.
```