```
unit LCRModel;

interface

{$M+}

uses Windows, dialogs, SysUtils, Classes,
     LCRConfig, LCRPWSRecords, LCRCostVars, LCRCosts,
LCRGlobals,CostingSteps,LCRBenefits,
     LCRMetricCollector, LCRResultsFile, SafewaterUncertBucket,System.Diagnostics,
     MtxVec, AbstractMtxVec, Math387, VLSSystemData, StateSchoolSampData,
LCRMicroOut;

type
  TTellGUIProc = procedure(Msg : string; var Stop : boolean) of object;

  TLCRThread = class;

  TLCRModel = class
  private
    fSW : TStopWatch;
    fInitialized : boolean;
    fConfigFileName : string;
    fOnModelDone : TNotifyEvent;
    FOnProgress : TTellGUIProc;
    InThread : TLCRThread;
    StopRequest : boolean;

    fResultsFile : TLCRResultsFile;
    fMicroOutput : TLCRMicroOut;

    strLeadConcentrationsS,strLeadConcentrationsB : TBufferedFileStream;

    DummyProb: double;

    procedure ModelFinished(Sender: TObject);
    procedure RunModel;
    procedure Deinitialize;
    function GetEP(const SW,SS : integer) : integer;
    procedure WritePWSBinCount;
    procedure SetCostingData(aPWS : TPWSRecordObj; var AddCostingData :
TAddCostGenRec);
  public
    PWS : TPWSRecords;
    PWSLoadRate : integer;
    MicroOutput : boolean;

    Config: TLCRConfig;
    Costs: TLCRCosts;
    BenefitsCollector: TBenefitsCollector;

    Uncertainty: TUncertaintyStudy;
    Outputs: TMetricList;

    StartLoop,EndLoop : integer;
    Categories,CategoriesMN : TCategoryList;  //output file

    //Metrics calculated in model object
    BCRatio,TotalNetBenefits,AnnNetBenefits : double;

    constructor Create(aConfigFileName : string); overload;
    constructor Create(aConfig: TLCRConfig); overload;
    destructor Destroy; override;

    function Initialize(AStream: TStream; userpath : string) : boolean;

    procedure Run;
    procedure RunInThread;
    procedure StopModel;
    procedure SaveOutput;
  published
    property OnModelDone : TNotifyEvent read FOnModelDone write FOnModelDone;
    property OnProgress: TTellGUIProc read FOnProgress write FOnProgress;
  end;

  TLCRThread = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  public
    Model : TLCRModel;
  end;

implementation

{ TLCRModel }

constructor TLCRModel.Create(aConfigFileName: string);
begin
  InThread := nil;
  OnModelDone := nil;
  fInitialized := false;
  fConfigFileName := aConfigFileName;

  Config := TLCRConfig.Create;
  Config.Load(fConfigFileName);

  PWSLoadRate := 1;
  StartLoop := 0;
  EndLoop := 0;
end;

constructor TLCRModel.Create(aConfig: TLCRConfig);
begin
  InThread := nil;
  OnModelDone := nil;
  fInitialized := false;

  Config := aConfig;

  PWSLoadRate := 1;
  StartLoop := 0;
  EndLoop := 0;
end;

procedure TLCRModel.Deinitialize;
begin
  PWS.Free;
  Outputs.Free;
  Uncertainty.Free;
  Costs.Free;
  BenefitsCollector.Free;
  Categories.Free;
  CategoriesMN.Free;
  fResultsFile.Destroy;
  fMicroOutput.Free;

  if Assigned(strLeadConcentrationsS) then
    strLeadConcentrationsS.Free;
  if Assigned(strLeadConcentrationsB) then
    strLeadConcentrationsB.Free;

end;

destructor TLCRModel.destroy;
begin
  if fInitialized then Deinitialize;
  Config.Free;
  inherited;
end;

function TLCRModel.GetEP(const SW, SS : integer): integer;
var i : integer;
```

```
    r,c : double;
begin
  Result:=0;
  r:=Random;
  c:=0;
  for i:=1 to Config.YearsOfAnalysis 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 TLCRModel.Initialize(AStream: TStream; userpath: string): boolean;
var
  sLine: string;
  BFile,SFile : string;
begin
  Result := true;

  StartLoop := 0;
  EndLoop := Config.NumberOfTrials;
  fInitialized := true;
  fMicroOutput :=
TLCRMicroOut.create(MicroOutput,UserPath+Config.RunName+'_Micro\');

  Config.ModelPreCalcs;

  fResultsFile := TLCRResultsFile.Create(UserPath+Config.RunName+'.swr',
                                        Config.YearsOfAnalysis, 1);

  Uncertainty := TUncertaintyStudy.Create(Config.NumberOfTrials);
  Config.PopulateUncertainty(Uncertainty);

  PWS := TPWSRecords.Create(Config);

  BFile:='';
  SFIle:='';
  if not Config.RunOptionOnly then BFile := Config.BasePWSDataFile;
  if not Config.RunBaselineOnly then SFile := Config.ScenPWSDataFile;
  PWS.OpenFromCSVPair(BFile,SFile,PWSLoadRate);

  InitPreCalcDR(Config.DiscountRate);

  Outputs := TMetricList.create(Config);
  Outputs.ResultsFile := fResultsFile;
  strLeadConcentrationsS := TBufferedFileStream.Create(UserPath+Config.RunName +
'_S_LeadConcentrations.csv', fmCreate, 4096);
  strLeadConcentrationsB := TBufferedFileStream.Create(UserPath+Config.RunName +
'_B_LeadConcentrations.csv', fmCreate, 4096);

  sLine := 'PWSID,Size,Source,Weight,' +
           'Population,Year,' +
           'G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14,G15,G16,Sum(G1-G16),' +

'LSL,CCT,PopPerConnection,NumLSLReplaced,pp_lslr_partial,pp_lslr_paper,cct_adjust_yr
,cct_install_yr,' +
           'perc_lsl,fAdjust_CCT,fInstall_CCT,num_lsl_remain,partial_cct_level' +
sLineBreak;

  strLeadConcentrationsS.WriteBuffer(sLine[1], Length(SLine)*SizeOf(Char));
  strLeadConcentrationsB.WriteBuffer(sLine[1], Length(SLine)*SizeOf(Char));

  Costs := TLCRCosts.Create(Config, Outputs, Uncertainty);
  Costs.strLeadConcentrationsB := strLeadConcentrationsB;
  Costs.strLeadConcentrationsS := strLeadConcentrationsS;

  BenefitsCollector:=TBenefitsCollector.create(Config, Outputs, Uncertainty);

  DummyProb := 1;
  Outputs.AddOutputMetric(@AnnNetBenefits,@DummyProb,nil,'Annual Net
Benefits',mtNetBenefits,False,False,False,Config.OptionName,0,false,true);
  Outputs.AddOutputMetric(@BCRatio,@DummyProb,nil,'Benefit Cost
Ratio',mtNetBenefits,False,False,False,Config.OptionName,0,False,True);

  Outputs.DoneAddingOutputs;


Categories:=TCategoryList.create(Config,Outputs,ChangeFileExt(fConfigFileName,'.swo'
));

CategoriesMN:=TCategoryList.create(Config,Outputs,ChangeFileExt(fConfigFileName,'.sw
omn'));
end;

procedure TLCRModel.ModelFinished(Sender: TObject);
begin
  if Assigned(FOnModelDone) then FOnModelDone(Self);
end;

procedure TLCRModel.Run;
begin
  InThread := nil;
```

```
    RunModel;
  ModelFinished(nil);
end;

procedure TLCRModel.RunInThread;
begin
  InThread := TLCRThread.Create(true);
  InThread.Model := self;
  InThread.FreeOnTerminate := true;
  InThread.OnTerminate := ModelFinished;
  InThread.Start;
end;

procedure TLCRModel.SetCostingData(aPWS : TPWSRecordObj; var AddCostingData :
TAddCostGenRec);
begin
  fillchar(AddCostingData,SizeOf(AddCostingData),0);
  if not Assigned(aPWS) then exit;
  AddCostingData.Bin := aPWS.Bin;
  AddCostingData.Small_Correct := aPWS.Small_Correct;
  AddCostingData.Num_Proxies := aPWS.Num_Proxies;
end;

procedure TLCRModel.RunModel;
var
  aPWS: TPWSRecordObj;
  i, samp, vloop: integer;
  CatCnt, UseRandSeed: integer;
  CatMembership : TStringList;
  OccConc: TVec;
  TC : DWORD;
  ev : int64;
  VLSSystemData: TVLSSystemData;
  stop : boolean;

  StateSchoolSampData: TStateSchoolSampData;

  slProxies: TStringList;
  sLine2: string;
  stateabb: string;
  SchoolSampData: TSchoolSampDataRec;
begin
  fSW := TStopWatch.StartNew;
  StopRequest := false;

  // comment the following line out to get the same results every run
  // uncomment to reset the random seed for each run
  //Randomize;
  RandSeed := 1;

  CatMembership := TStringList.Create;
  //TC:=gettickcount;
  try
    createit(OccConc);
    OccConc.Size(1);
    OccConc[0]:=1;

    // State costs: these do not vary by system attribute
    Costs.StateCosts;

    VLSSystemData := TVLSSystemData.Create;
    StateSchoolSampData := TStateSchoolSampData.Create;
    StateSchoolSampData.LoadSchoolSampData;

    for samp := StartLoop to EndLoop do begin
      if samp = 0 then
        Uncertainty.ResetValues;

      for vloop := 1 to Config.NumberOfVLoops do begin
        i := 0;

        while PWS.Next do begin

          if not Config.RunBaselineOnly then begin
            aPWS := PWS.CurScenPWS;
            if UserSeeds then UseRandSeed:=PWS.UserRandSeedS;
          end else begin
            aPWS := PWS.CurBasePWS;
            if UserSeeds then UseRandSeed:=PWS.UserRandSeedB;
          end;
          if not aPWS.RunIt then continue;

          if Config.RunSysType = 'CWS' then
          begin
            if aPWS.SystemType <> sysCWS then continue;
          end
          else
          if Config.RunSysType = 'NTNCWS' then
          begin
            if aPWS.SystemType <> sysNTNC then continue;
          end;

Costs.CostingData.NumProxies:=aPWS.Num_Proxies;

          Inc(i);
          if (i mod 10 = 1) then begin
```

```
            if Assigned(FOnProgress) then begin
              FOnProgress('samp:'+samp.ToString+', v:'+vloop.ToString+',
on:'+i.tostring+', tot:',stop);
              if stop then break;
            end;
          end;



          if Assigned(PWS.CurScenPWS) then
            Costs.CostingData.fScenVars:=PWS.CurScenPWS.fVars;
          if Assigned(PWS.CurBasePWS) then
            Costs.CostingData.fBaseVars:=PWS.CurBasePWS.fVars;

          Costs.CostingData.PWSid := String(aPWS.PWSId);
          Costs.CostingData.SystemSize:=Integer(aPWS.SystemSize) + 1;   // add one
because values in database begin with 1
          Costs.CostingData.SourceWater:=Integer(aPWS.SourceWater) + 1; // add one
because values in database begin with 1
          Costs.CostingData.SystemType:=Integer(aPWS.SystemType) + 1; // add one
because values in database begin with 1
          Costs.CostingData.Ownership:=Integer(aPWS.Ownership);
          Costs.CostingData.Population:=aPWS.Population;
          Costs.CostingData.InflatedPops:=@aPWS.InflatedPopulation;
          Costs.CostingData.LSL := aPWS.LSL; //0=No 1=Yes
          Costs.CostingData.CCT := aPWS.CCT; //0=No 1=Yes
          Costs.CostingData.SamplingWeight:=aPWS.SamplingWeight;
          Costs.CostingData.AvgRevenue:=aPWS.AvgRevenue ;

          Costs.CostingData.EntryPoints:=aPWS.NumberEPs;

          if Costs.CostingData.EntryPoints = -1 then Costs.CostingData.EntryPoints
:= 0;

          Costs.CostingData.Connections := aPWS.Connections;
          Costs.CostingData.First_ale := aPWS.First_ale; //0=No 1=Yes
          Costs.CostingData.CostCapital := aPWS.CostCapital;

          Costs.CostingData.NumberLSLs := aPWS.NumberLSLs;
          Costs.CostingData.CCTPO4 := aPWS.CCTPO4;
          Costs.CostingData.CCTPH := aPWS.CCTPH;
          Costs.CostingData.CCTBoth := aPWS.CCTBoth;
          Costs.CostingData.BaselinePO4Dose := aPWS.BaselinePO4Dose;
          Costs.CostingData.BaselinePH_wPh := aPWS.BaselinePH_wPh;
          Costs.CostingData.BaselinePH_woPh := aPWS.BaselinePH_woPh;
          Costs.CostingData.BaselinePH_woCCT := aPWS.BaselinePH_woCCT;
          Costs.CostingData.BaselinePH_wPO4Ph := aPWS.BaselinePH_wPO4Ph;
```

```
//*************************************
          //set the flows.  This should be done yearly.  Address when we are
ready...
          aPWS.GetFlows(Costs.CostingData.EntryPoints,0,Config,
                        Costs.CostingData.AFlowEP,Costs.CostingData.DFlowEP);
          Costs.CostingData.PWSAnnualRevenue:=aPWS.PWSAnnualRevenue;

          if Costs.CostingData.SystemSize = 9 then
          begin
            VLSSystemData.GetSystemData(Costs.CostingData.PWSid);
            Costs.CostingData.Connections := VLSSystemData.Connections;

            if VLSSystemData.NumberLSLs >= 0 then
              Costs.CostingData.NumberLSLs := VLSSystemData.NumberLSLs;

            Costs.CostingData.P90_base := VLSSystemData.P90_base;
            if Costs.CostingData.NumberLSLs > 0 then
              Costs.CostingData.LSL := 1
            else
              Costs.CostingData.LSL := 0;
          end;

          SetCostingData(PWS.CurBasePWS,Costs.BAddCostingData);
          SetCostingData(PWS.CurScenPWS,Costs.SAddCostingData);

          SchoolSampData :=
StateSchoolSampData.GetStateSchoolSampData(copy(aPWS.PWSId,1,2));

          // PWS level costs
          fMicroOutput.AddPWS(Costs.CostingData);
          Costs.GenerateCosts(UseRandSeed, slProxies, SchoolSampData);

          BenefitsCollector.DoDebugOut:=Random(200)=1;
          BenefitsCollector.NewBenBins:=True;

          BenefitsCollector.GenerateBenefits(Costs, aPWS.Num_Proxies>0);


          CatMembership.CommaText:=aPWS.CategoryMembership;

          if aPWS.Num_Proxies=0 then begin
            //May be unnessary...
            Outputs.CollectWeightedObs;
            Outputs.ContamLevelDone(1);
            Outputs.ApplyOccDist(OccConc,1e-6);
            for CatCnt:=0 to CatMembership.Count-1 do begin
              if Samp=0 then
```

```
CategoriesMN.CollectVariability(strtoint(CatMembership.Strings[CatCnt]),Costs.Costin
gData.SamplingWeight)
              else

Categories.CollectVariability(strtoint(CatMembership.Strings[CatCnt]),Costs.CostingD
ata.SamplingWeight);
            end;
          end else begin
            aPWS.Cost:=Costs.TotalCostCap;
          end;

          if StopRequest then break;

        end; // end PWS loop

        if stop then break;
        WritePWSBinCount;
      end; // Variability loop

      if Samp=0 then
        CategoriesMN.CollectUncertainty
      else
        Categories.CollectUncertainty;
      if stop then break;
    end; // end sample loop

    VLSSystemData.Free;
    StateSchoolSampData.Free;
  except
    on E : Exception do
    begin
       StateSchoolSampData.Free;
       VLSSystemData.Free;
       CatMembership.Free;
       FreeIt(OccConc);
       ShowMessage('Exception class name = '+E.ClassName);
       ShowMessage('Exception message = '+E.Message);
     end;
  end;

  CatMembership.Free;
  FreeIt(OccConc);

  Config.Log.TimeText:='Run time (minutes):' + fSW.Elapsed.TotalMinutes.ToString;

  SaveOutput;
  ev:=costs.GetTotalEvaluations;
  Config.Log.Text:='Total Cost Parser Calls:'+ev.ToString;
  ev:=costs.GetTotalCompiledEvaluations;
  Config.Log.Text:='Total Compiled Cost Calls:'+ev.ToString;
  Config.Log.WriteIt(changefileext(fConfigFileName,'.log'));
end;

procedure TLCRModel.SaveOutput;
begin
  Categories.GenerateFinalOutput;
  Categories.SaveCategoryOutput;
  CategoriesMN.GenerateFinalOutput;
  CategoriesMN.SaveCategoryOutput;
end;

procedure TLCRModel.StopModel;
begin
  if Assigned(InThread) then InThread.Terminate;
  StopRequest:=True;
end;

procedure TLCRModel.WritePWSBinCount;
var
  st,i,j,k: integer;
  sLine: string;
  SL: TStringList;
begin
  if Config.RunDifference then exit;
  SL := TStringList.Create;

  sLine :=
'SystemType'+chr(9)+'Year'+chr(9)+'Size'+chr(9)+'Source'+chr(9)+'Bin1'+chr(9)+'Bin2'
+chr(9)+'Bin3';
  SL.Add(sLine);

  for st := 1 to 2 do
  for i := 1 to Config.YearsOfAnalysis do
    for j := 1 to 9 do
      for k := 1 to 2 do
      begin
         sLine :=
st.ToString+chr(9)+i.ToString+chr(9)+j.ToString+chr(9)+k.ToString+chr(9)+
                  Config.PWSBinCount[st,i,j,k,1].ToString+chr(9)+
                  Config.PWSBinCount[st,i,j,k,2].ToString+chr(9)+
                  Config.PWSBinCount[st,i,j,k,3].ToString;
         SL.Add(sLine);
      end;

  SL.SaveToFile(UserPath + '\' + Config.RunName + '_BinCounts.tab');
  SL.Free;
```

```
end;

{ TLCRThread }

procedure TLCRThread.Execute;
begin
  inherited;
  try
    Model.RunModel;
  except
    //on e:Exception do
    //  Model.ProgressMsg:='Grand exception:'+e.Message;
  end;
end;

end.
```