```
unit LCRConfig;

interface

uses Types, Classes, Contnrs, SysUtils, Math, LCRGlobals, BigIni,
     SafewaterUncertBucket;

type
  TRunLog=class
  private
     fLog : TStringList;
     fExtras : TStringList;
     procedure put_log_txt(txt:string);
     procedure put_log_txt_nostamp(txt:string);
     procedure put_log_txt_strings(txt:TstringList);
  public
     constructor Create;
     destructor Destroy; override;

     procedure AddExtra(Tag,HTML : string);
     function GetExtra(Tag : string) : string;

     procedure LoadFromStream(Strm : TStream);
     procedure SaveToStream(Strm : TStream);
     function HTMLLog : string;
     procedure WriteIt(aFile : string);


     property TimeText : String Write put_log_txt;
     property Text : String Write put_log_txt_nostamp;
     property Strings : TStringList Write put_log_txt_strings;
  end;

  TLCRConfig = class
  private
    fYearsOfAnalysis: integer;
    fYearsOfOutput: integer;
    fDiscStr: string;

    procedure ExpectedEP;
    procedure SetDR(value: string);
    procedure DefaultFlowVars;
  public
    RunName: string;
    OptionName: string;
    RunDescription: string;
    FullLog: boolean;  //set to turn on off mucho logging
    MeanUncertLevel: boolean;      //Run analysis only at the means of uncertain
```

```
params
    Seed           : longint;      //Random Number Seed
    CCTCostEquationLevel: string;

    RunOptionOnly: boolean;
    RunBaselineOnly: boolean;
    RunDifference: boolean;
    RunNoBLAveraging : boolean;
    NoICRCosts: boolean;
    VariabilityRun: boolean;
    OutputLeadBins, Debug,CCTToFull : boolean;
    SmallSystemFlexibility: boolean;
    BenByYear : boolean;

    fBaseYear: Integer;
    DiscountRate: double;
    DiscRates: array of double; //Should not be set explicity - set through List
properties

    OCategoryDefs : TObjectList;
    OccMap: TStringList;
    UseCompiledCost, BenefitsAll : boolean;

    Log: TRunLog;

    PWSDataFile: string;
    BasePWSDataFile,ScenPWSDataFile : string;
    BaseCostSteps,ScenCostSteps,
    BaseVarData,ScenVarData : string;
    OccurrenceMapFileXL, EPProbsXL: string;
    CCTCostEquations: string;

    StateDataArray: TStateDataArray;

    NumberOfTrials, NumberOfVLoops : integer;
    QuantilePoints : TDoubleDynArray; //array of quantile points to store across
uncertainty
    QuantilePrecision : double;
    UncQuantilePoints : TDoubleDynArray; //array of quantile points to output for
uncertainty

    SystemType: TSystemType;

    EntryPointProbs : array[0..1,0..7,1..50] of double;   //Number of entry points
and probabilities
    ExpectedEntryPoint : array[0..1,0..7] of double;   //Number of entry points and
probabilities
    PWSCapitalCost : array[1..2,0..8] of double;
    PWSBinCount : array[1..2,1..60,1..9,1..2,1..3] of double;

    FlowVars, DFlowVars : array[1..2,1..2,1..2] of TSWVar;
    LossRate: TSWVar;

    DefaultPopPct : TSubPopArray;

    SmallProxyPop: integer;
    PWS90PctBp1: integer;
    PWS90PctBp2: integer;

    SchoolOption: string;
    RunSysType: string;

    PWSRevenue : array[1..2,0..8] of double;
    RevenueYear: integer;
    RevenueInflationFactor: double;

    ChildBLSens : integer; //0=normal, 1=1.15 2 = 1.75;
    IQValueSens : integer; //0=normal, 1=alt values;
    IQDRSens : integer; //0=normal, 1=naacs low dose linerization 2= naacs without
low dose linerization
    CVDDRSens : integer; //0=normal; 1=Lanphear
    VolLeadProg: integer;

    constructor Create;
    destructor Destroy; override;

    procedure Load(FileName : string);
    procedure Save(FileName : string);

    procedure RenumberCategories;
    function ConnectExcelOccurrenceMap: boolean;
    function ConnectExcelEPProbs : boolean;
    function GetOccRegion(State: string): integer;
    procedure PopulateUncertainty(Unc : TUncertaintyStudy);
    procedure ModelPreCalcs;
    function GetCostOfCapital(Ownership : TOwnership; SysSize : double): double;
    function GetAvgRevenue(Ownership : TOwnership;  SysSize : double): double;

    property YearsOfAnalysis: Integer read fYearsOfAnalysis;
    property YearsOfOutput: Integer read fYearsOfOutput;
    property DiscRateList: string read fDiscStr write SetDR;

    procedure GetFlowsEp(const EP, Ownership, SourceWater, Population: integer;
                         var AFlowEp, DFlowEp: double);
    procedure GetFlows(Ownership, SourceWater, Population: integer;
                       var AFlow, DFlow: double);
```

```
  end;

implementation

uses VCL.FlexCel.Core, FlexCel.XLSAdapter;

{ TLCRConfig }

function TLCRConfig.ConnectExcelEPProbs: boolean;
var
  xls: TExcelFile;
  r: integer;
  FileName: string;
  i,sz,sw: integer;
  tot : double;
begin
  Result:=True;
  if SystemType=sysCWS then
    EPProbsXL:=DataPath+'CWSEntryPointProbs.xls'
  else
    EPProbsXL:=DataPath+'NTNCWSEntryPointProbs.xls';

  FileName := EPProbsXL;
  if FileName = '' then
    exit;

  xls := TXlsFile.Create;
  xls.Open(FileName);

  fillchar(EntryPointProbs,SizeOF(EntryPointProbs),0);

  for r := 2 to xls.RowCount do
  begin
    sw:=Integer(StrToTSourceWater(xls.GetStringFromCell(r, 2)));
    sz:=Integer(StrToTSystemSize(xls.GetStringFromCell(r, 3)));
    tot:=0;
    for i:=4 to 53 do begin
       EntryPointProbs[sw,sz,i-3] := xls.GetCellValue(r,i).AsNumber;
       tot:=tot+EntryPointProbs[sw,sz,i-3];
    end;
    if abs(tot-1)>0.0001 then begin
      for i:=4 to 53 do begin
         EntryPointProbs[sw,sz,i-3]:=EntryPointProbs[sw,sz,i-3]/tot;
      end;
    end;
  end;

  xls.Free;
  ExpectedEP;
end;

function TLCRConfig.ConnectExcelOccurrenceMap: boolean;
var
  xls: TExcelFile;
  r: integer;
  FileName: string;
  cVal: string;
  i, OccRegion: integer;
begin
  Result:=True;
  FileName := OccurrenceMapFileXL;
  if FileName = '' then
    exit;

  OccMap.Clear;

  xls := TXlsFile.Create;
  xls.Open(FileName);

  for r := 2 to xls.RowCount do
  begin
    cVal := xls.GetStringFromCell(r, 1);
    i := OccMap.Add(cVal);
    OccRegion := trunc(xls.GetCellValue(r,2).AsNumber);
    OccMap.Objects[i] := TObject(OccRegion);
  end;

  xls.Free;
end;

constructor TLCRConfig.Create;
var
  i, l, s, o : integer;
  TP: TPicked;
begin
  PWSDataFile := DataPath+'SDWIS CWS 2016 Q3.xlsx';
  BasePWSDataFile := DataPath+'Baseline_NoRep.csv';
  ScenPWSDataFile := DataPath+'OWNoRep.csv';

  OccurrenceMapFileXL := DataPath+'State_DefaultRegion.xls';
  EPProbsXL := DataPath+'CWSEntryPointProbs.xls';

  BaseCostSteps:=DataPath+'Baseline_costing_inputs.xlsx';
  ScenCostSteps:=DataPath+'CWS_NDWAC_costing_inputs.xlsx';
  BaseVarData:=DataPath+'Baseline_DataRequest.accdb';
  ScenVarData:=DataPath+'OW_DataRequest_VLS.accdb';

  CCTCostEquations := DataPath + 'CCTCostEquations.xlsx';

// Set Years of Analysis
  fBaseYear:=2016;
  fYearsOfAnalysis := 60;
  fYearsOfOutput := 35;

  SystemType := sysCWS;
  QuantilePrecision:=0.01;

  PWS90PctBp1 := 10;
  PWS90PctBp2 := 15;
  SmallProxyPop := 3300;

  RunName:='Run Name';
  OptionName:='Option Name';
  RunOptionOnly := false;
  RunBaselineOnly := false;
  RunDifference := true;
  RunNoBLAveraging := false;
  NoICRCosts := true;
  Log:=TRunLog.Create;
  OutputLeadBins := false;
  SmallSystemFlexibility := true;
  CCTCostEquationLevel := '';
  UseCompiledCost := false;
  BenefitsAll := false;
  BenByYear := false;

  fillchar(PWSBinCount, SizeOf(PWSBinCount), 0);

  NumberOfTrials := 0;
  VariabilityRun := true;
  NumberofVLoops:=1;
  MeanUncertLevel:=False;
  Seed:=0;

  setlength(QuantilePoints,5);
  QuantilePoints[0]:=0.1;
  QuantilePoints[1]:=0.25;
  QuantilePoints[2]:=0.51;
  QuantilePoints[3]:=0.75;
  QuantilePoints[4]:=0.90;

  setlength(UncQuantilePoints,4);
```

```
UncQuantilePoints[0]:=0.01;
UncQuantilePoints[1]:=0.05;
UncQuantilePoints[2]:=0.95;
UncQuantilePoints[3]:=0.99;

fDiscStr := '0.03,0.07';
SetDR(fDiscStr);
DiscountRate := 0.03;

PWSRevenue[1,0]:=9.17;  PWSRevenue[1,1]:=4.56;
PWSRevenue[1,2]:=4.24;  PWSRevenue[1,3]:=3.63;
PWSRevenue[1,4]:=3.39;  PWSRevenue[1,5]:=4.01;
PWSRevenue[1,6]:=3.41;  PWSRevenue[1,7]:=2.39;

PWSRevenue[1,8]:=2.39;

PWSRevenue[2,0]:=4.19;  PWSRevenue[2,1]:=5.21;
PWSRevenue[2,2]:=6.61;  PWSRevenue[2,3]:=5.61;
PWSRevenue[2,4]:=5.17;  PWSRevenue[2,5]:=3.2;
PWSRevenue[2,6]:=4.03;  PWSRevenue[2,7]:=2.41;

PWSRevenue[2,8]:=2.41;

RevenueYear := 2006;
RevenueInflationFactor := 1.03;

IQValueSens :=0;
IQDRSens :=0;
CVDDRSens :=0;
ChildBLSens:=0;
VolLeadProg := 0;

DefaultFlowVars;

//default categories
OCategoryDefs:=TObjectList.Create(false);

OccMap := TStringList.Create;

for s:=0 to 1 do begin
  TP:=Tpicked.Create;
  TP.SysType[s+1]:=True;
  TP.name:=TP.Getname;
  OCategoryDefs.Add(TP);
end;

for o:=0 to 1 do begin
  TP:=Tpicked.Create;
  TP.Owner[o+1]:=True;
  TP.name:=TP.Getname;
  OCategoryDefs.Add(TP);
end;

for o:=0 to 1 do
  for s:=0 to 1 do
    for l:=0 to 1 do
      for i:=0 to 8 do begin
         TP:=Tpicked.Create;
         TP.Source[l+1]:=True;
         TP.Size[i+1]:=True;
         TP.SysType[s+1]:=True;
         TP.Owner[o+1]:=True;
         TP.name:=TP.Getname;
         OCategoryDefs.Add(TP);
      end;

RenumberCategories;

// length set in SafeWaterGlobals
StateDataArray[0].StateCode  := 'AL';
StateDataArray[1].StateCode  := 'AK';
StateDataArray[2].StateCode  := 'AS';
StateDataArray[3].StateCode  := 'AZ';
StateDataArray[4].StateCode  := 'AR';
StateDataArray[5].StateCode  := 'CA';
StateDataArray[6].StateCode  := 'CO';
StateDataArray[7].StateCode  := 'CT';
StateDataArray[8].StateCode  := 'DE';
StateDataArray[9].StateCode  := 'DC';
StateDataArray[10].StateCode := 'FL';
StateDataArray[11].StateCode := 'GA';
StateDataArray[12].StateCode := 'GU';
StateDataArray[13].StateCode := 'HI';
StateDataArray[14].StateCode := 'ID';
StateDataArray[15].StateCode := 'IL';
StateDataArray[16].StateCode := 'IN';
StateDataArray[17].StateCode := 'IA';
StateDataArray[18].StateCode := 'KS';
StateDataArray[19].StateCode := 'KY';
StateDataArray[20].StateCode := 'LA';
StateDataArray[21].StateCode := 'ME';
StateDataArray[22].StateCode := 'MD';
StateDataArray[23].StateCode := 'MH';
StateDataArray[24].StateCode := 'MA';
StateDataArray[25].StateCode := 'MI';
StateDataArray[26].StateCode := 'FM';
```

```
StateDataArray[27].StateCode := 'MN';
StateDataArray[28].StateCode := 'MS';
StateDataArray[29].StateCode := 'MO';
StateDataArray[30].StateCode := 'MT';
StateDataArray[31].StateCode := 'NE';
StateDataArray[32].StateCode := 'NV';
StateDataArray[33].StateCode := 'NH';
StateDataArray[34].StateCode := 'NJ';
StateDataArray[35].StateCode := 'NM';

StateDataArray[36].StateCode := 'NN';

StateDataArray[37].StateCode := 'NY';
StateDataArray[38].StateCode := 'NC';
StateDataArray[39].StateCode := 'ND';
StateDataArray[40].StateCode := 'MP';
StateDataArray[41].StateCode := 'OH';
StateDataArray[42].StateCode := 'OK';
StateDataArray[43].StateCode := 'OR';
StateDataArray[44].StateCode := 'PW';
StateDataArray[45].StateCode := 'PA';
StateDataArray[46].StateCode := 'PR';
StateDataArray[47].StateCode := 'RI';
StateDataArray[48].StateCode := 'SC';
StateDataArray[49].StateCode := 'SD';
StateDataArray[50].StateCode := 'TN';
StateDataArray[51].StateCode := 'TX';

StateDataArray[52].StateCode := '01';
StateDataArray[53].StateCode := '02';
StateDataArray[54].StateCode := '03';
StateDataArray[55].StateCode := '04';
StateDataArray[56].StateCode := '05';
StateDataArray[57].StateCode := '06';
StateDataArray[58].StateCode := '07';
StateDataArray[59].StateCode := '08';
StateDataArray[60].StateCode := '09';
StateDataArray[61].StateCode := '10';

StateDataArray[62].StateCode := 'UT';
StateDataArray[63].StateCode := 'VT';
StateDataArray[64].StateCode := 'VA';
StateDataArray[65].StateCode := 'VI';
StateDataArray[66].StateCode := 'WA';
StateDataArray[67].StateCode := 'WV';
StateDataArray[68].StateCode := 'WI';
StateDataArray[69].StateCode := 'WY';
```

```
  for i := 0 to 69 do
    StateDataArray[i].PWSCount := 0;

  PWSCapitalCost[1,0]:=0.6;  PWSCapitalCost[1,1]:=3.9;
  PWSCapitalCost[1,2]:=3.2;  PWSCapitalCost[1,3]:=3.9;
  PWSCapitalCost[1,4]:=4.1;  PWSCapitalCost[1,5]:=4.2;
  PWSCapitalCost[1,6]:=3.7;  PWSCapitalCost[1,7]:=4.3;
  PWSCapitalCost[1,8]:=4.3;

  PWSCapitalCost[2,0]:=5.4;  PWSCapitalCost[2,1]:=7.3;
  PWSCapitalCost[2,2]:=6.5;  PWSCapitalCost[2,3]:=5.9;
  PWSCapitalCost[2,4]:=5.4;  PWSCapitalCost[2,5]:=5.4;
  PWSCapitalCost[2,6]:=3.6;  PWSCapitalCost[2,7]:=6.2;
  PWSCapitalCost[2,8]:=6.2;

  for i:=0 to 4 do DefaultPopPct[1,i]:= 0.035581077/5;
  for i:=5 to 9 do DefaultPopPct[1,i]:= 0.034334366/5;
  for i:=10 to 14 do DefaultPopPct[1,i]:= 0.036769403/5;
  for i:=15 to 19 do DefaultPopPct[1,i]:= 0.036072206/5;
  for i:=20 to 24 do DefaultPopPct[1,i]:= 0.035347534/5;
  for i:=25 to 29 do DefaultPopPct[1,i]:= 0.033747072/5;
  for i:=30 to 34 do DefaultPopPct[1,i]:= 0.033832934/5;
  for i:=35 to 44 do DefaultPopPct[1,i]:= 0.073734571/10;
  for i:=45 to 54 do DefaultPopPct[1,i]:= 0.070595468/10;
  for i:=55 to 64 do DefaultPopPct[1,i]:= 0.048257695/10;
  for i:=65 to 74 do DefaultPopPct[1,i]:= 0.029076197/10;
  for i:=75 to 84 do DefaultPopPct[1,i]:= 0.018577032/10;
  DefaultPopPct[1,85]:=0.004365207;

  for i:=0 to 4 do DefaultPopPct[2,i]:=0.034011526/5;
  for i:=5 to 9 do DefaultPopPct[2,i]:= 0.032795725/5;
  for i:=10 to 14 do DefaultPopPct[2,i]:= 0.035086514/5;
  for i:=15 to 19 do DefaultPopPct[2,i]:= 0.03489075/5;
  for i:=20 to 24 do DefaultPopPct[2,i]:= 0.034739633/5;
  for i:=25 to 29 do DefaultPopPct[2,i]:= 0.033231902/5;
  for i:=30 to 34 do DefaultPopPct[2,i]:= 0.034200422/5;
  for i:=35 to 44 do DefaultPopPct[2,i]:= 0.075153006/10;
  for i:=45 to 54 do DefaultPopPct[2,i]:= 0.073514765/10;
  for i:=55 to 64 do DefaultPopPct[2,i]:= 0.053182721/10;
  for i:=65 to 74 do DefaultPopPct[2,i]:= 0.03408365/10;
  for i:=75 to 84 do DefaultPopPct[2,i]:= 0.02599548/10;
  DefaultPopPct[2,85]:=0.00882658;

end;

procedure TLCRConfig.DefaultFlowVars;
var i,j,k : integer;
begin
```

```
fillchar(FlowVars,Sizeof(FlowVars),0);
fillchar(DFlowVars,Sizeof(DFlowVars),0);
for i:=1 to 2 do
  for j:=1 to 2 do
    for k:=1 to 2 do begin
      FlowVars[i,j,k].UpperBound:=9e99;
      DFlowVars[i,j,k].UpperBound:=9e99;
      FlowVars[i,j,k].LowerBound:=-9e99;
      DFlowVars[i,j,k].LowerBound:=-9e99;
      FlowVars[i,j,k].CensorValue:=-9e99;
      DFlowVars[i,j,k].CensorValue:=-9e99;
    end;


fillchar(LossRate,sizeof(LossRate),0);
LossRate.Name := 'LossRate';
LossRate.PointValue:=0.1;
LossRate.V:=LossRate.PointValue;
LossRate.Parm1:=0;
LossRate.Parm2:=0.2;
LossRate.Distribution:=dUniform;

FlowVars[2,2,1].Name:='PrivSurfA';
FlowVars[2,2,1].PointValue:=0.09036;
FlowVars[2,2,1].V:=FlowVars[2,2,1].PointValue;
FlowVars[2,2,1].Distribution:=dNone;
FlowVars[2,2,1].Parm1 := FlowVars[2,2,1].PointValue;

FlowVars[2,2,2].Name:='PrivSurfB';
FlowVars[2,2,2].PointValue:=1.03338;
FlowVars[2,2,2].V:=FlowVars[2,2,2].PointValue;
FlowVars[2,2,2].Distribution:=dNone;
FlowVars[2,2,2].Parm1 := FlowVars[2,2,2].PointValue;

FlowVars[1,2,1].Name:='PubSurfA';
FlowVars[1,2,1].PointValue:=0.14004;
FlowVars[1,2,1].V:=FlowVars[1,2,1].PointValue;
FlowVars[1,2,1].Distribution:=dNone;
FlowVars[1,2,1].Parm1 := FlowVars[1,2,1].PointValue;

FlowVars[1,2,2].Name:='PubSurfB';
FlowVars[1,2,2].PointValue:=0.99703;
FlowVars[1,2,2].V:=FlowVars[1,2,2].PointValue;
FlowVars[1,2,2].Distribution:=dNone;
FlowVars[1,2,2].Parm1 := FlowVars[1,2,2].PointValue;

FlowVars[2,1,1].Name:='PrivGroundA';
FlowVars[2,1,1].PointValue:=0.06670;
```

```
FlowVars[2,1,1].V:=FlowVars[2,1,1].PointValue;
FlowVars[2,1,1].Distribution:=dNone;
FlowVars[2,1,1].Parm1 := FlowVars[2,1,1].PointValue;

FlowVars[2,1,2].Name:='PrivGroundB';
FlowVars[2,1,2].PointValue:=1.06284;
FlowVars[2,1,2].V:=FlowVars[2,1,2].PointValue;
FlowVars[2,1,2].Distribution:=dNone;
FlowVars[2,1,2].Parm1 := FlowVars[2,1,2].PointValue;

FlowVars[1,1,1].Name:='PubGroundA';
FlowVars[1,1,1].PointValue:=0.08575;
FlowVars[1,1,1].V:=FlowVars[1,1,1].PointValue;
FlowVars[1,1,1].Distribution:=dNone;
FlowVars[1,1,1].Parm1 := FlowVars[1,1,1].PointValue;

FlowVars[1,1,2].Name:='PubGroundB';
FlowVars[1,1,2].PointValue:=1.05839;
FlowVars[1,1,2].V:=FlowVars[1,1,2].PointValue;
FlowVars[1,1,2].Distribution:=dNone;
FlowVars[1,1,2].Parm1 := FlowVars[1,1,2].PointValue;

DFlowVars[2,2,1].Name:='D_PrivSurfA';
DFlowVars[2,2,1].PointValue:=0.35674;
DFlowVars[2,2,1].V:=DFlowVars[2,2,1].PointValue;
DFlowVars[2,2,1].Distribution:=dNone;
DFlowVars[2,2,1].Parm1 := DFlowVars[2,2,1].PointValue;

DFlowVars[2,2,2].Name:='D_PrivSurfB';
DFlowVars[2,2,2].PointValue:=0.96188;
DFlowVars[2,2,2].V:=DFlowVars[2,2,2].PointValue;
DFlowVars[2,2,2].Distribution:=dNone;
DFlowVars[2,2,2].Parm1 := DFlowVars[2,2,2].PointValue;

DFlowVars[1,2,1].Name:='D_PubSurfA';
DFlowVars[1,2,1].PointValue:=0.59028;
DFlowVars[1,2,1].V:=DFlowVars[1,2,1].PointValue;
DFlowVars[1,2,1].Distribution:=dNone;
DFlowVars[1,2,1].Parm1 := DFlowVars[1,2,1].PointValue;

DFlowVars[1,2,2].Name:='D_PubSurfB';
DFlowVars[1,2,2].PointValue:=0.94573;
DFlowVars[1,2,2].V:=DFlowVars[1,2,2].PointValue;
DFlowVars[1,2,2].Distribution:=dNone;
DFlowVars[1,2,2].Parm1 := DFlowVars[1,2,2].PointValue;

DFlowVars[2,1,1].Name:='D_PrivGroundA';
DFlowVars[2,1,1].PointValue:=0.41682;
```

```
  DFlowVars[2,1,1].V:=DFlowVars[2,1,1].PointValue;
  DFlowVars[2,1,1].Distribution:=dNone;
  DFlowVars[2,1,1].Parm1 := DFlowVars[2,1,1].PointValue;

  DFlowVars[2,1,2].Name:='D_PrivGroundB';
  DFlowVars[2,1,2].PointValue:=0.96078;
  DFlowVars[2,1,2].V:=DFlowVars[2,1,2].PointValue;
  DFlowVars[2,1,2].Distribution:=dNone;
  DFlowVars[2,1,2].Parm1 := DFlowVars[2,1,2].PointValue;

  DFlowVars[1,1,1].Name:='D_PubGroundA';
  DFlowVars[1,1,1].PointValue:=0.54992;
  DFlowVars[1,1,1].V:=DFlowVars[1,1,1].PointValue;
  DFlowVars[1,1,1].Distribution:=dNone;
  DFlowVars[1,1,1].Parm1 := DFlowVars[1,1,1].PointValue;

  DFlowVars[1,1,2].Name:='D_PubGroundB';
  DFlowVars[1,1,2].PointValue:=0.95538;
  DFlowVars[1,1,2].V:=DFlowVars[1,1,2].PointValue;
  DFlowVars[1,1,2].Distribution:=dNone;
  DFlowVars[1,1,2].Parm1 := DFlowVars[1,1,2].PointValue;
end;


destructor TLCRConfig.Destroy;
var
  i: integer;
begin
  for i := 0 to OCategoryDefs.Count - 1 do
    TPicked(OCategorydefs.Items[i]).Free;
  OCategorydefs.Free;

  OccMap.Free;
  Log.Free;

  inherited;
end;

procedure TLCRConfig.ExpectedEP;
var i,j,k : integer;
begin
  for i:=0 to 1 do begin
    for j:=0 to 7 do begin
      ExpectedEntryPoint[i,j]:=0;
      for k:=1 to 50 do begin
        ExpectedEntryPoint[i,j]:=ExpectedEntryPoint[i,j]+EntryPointProbs[i,j,k]*k;
      end;
    end;
  end;
end;

function TLCRConfig.GetCostOfCapital(Ownership: TOwnership;
  SysSize: double): double;
var i : integer;
begin
  i:=Integer(Ownership)+1;
  if SysSize<100 then Result:=PWSCapitalCost[i,0] else
  if SysSize<=500 then Result:=PWSCapitalCost[i,1] else
  if SysSize<=3300 then Result:=PWSCapitalCost[i,2] else
  if SysSize<=10000 then Result:=PWSCapitalCost[i,3] else
  if SysSize<=50000 then Result:=PWSCapitalCost[i,4] else
  if SysSize<=100000 then Result:=PWSCapitalCost[i,5] else
  if SysSize<=500000 then Result:=PWSCapitalCost[i,6] else
  if SysSize<=1000000 then Result:=PWSCapitalCost[i,7] else
                         Result:=PWSCapitalCost[i,8];

  Result:=Result/100;
end;

function TLCRConfig.GetAvgRevenue(Ownership : TOwnership; SysSize : double): double;
var i : integer;
begin
  i:=Integer(Ownership)+1;
  if SysSize<100 then Result:=PWSRevenue[i,0] else
  if SysSize<=500 then Result:=PWSRevenue[i,1] else
  if SysSize<=3300 then Result:=PWSRevenue[i,2] else
  if SysSize<=10000 then Result:=PWSRevenue[i,3] else
  if SysSize<=50000 then Result:=PWSRevenue[i,4] else
  if SysSize<=100000 then Result:=PWSRevenue[i,5] else
  if SysSize<=500000 then Result:=PWSRevenue[i,6] else
  if SysSize<=1000000 then Result:=PWSRevenue[i,7] else
                         Result:=PWSRevenue[i,8];

  Result:=Result * intpower(RevenueInflationFactor,fBaseYear-RevenueYear);
end;


procedure TLCRConfig.GetFlowsEp(const EP, Ownership, SourceWater, Population:
integer;
                                var AFlowEp, DFlowEp: double);
var fAvgFlowA,fAvgFlowB,fDesignFlowA,fDesignFlowB,dtmp : double;
    fPopulation : double;
begin
  fAvgFlowA:=FlowVars[Ownership+1,SourceWater,1].V;
  fAvgFlowB:=FlowVars[Ownership+1,SourceWater,2].V;
  fDesignFlowA:=DFlowVars[Ownership+1,SourceWater,1].V;
  fDesignFlowB:=DFlowVars[Ownership+1,SourceWater,2].V;

  fPopulation := Population;

  AFlowEp := ( fAvgFlowA * Power(fPopulation,fAvgFlowB) * 0.001 ) / EP;
  dtmp := ( fDesignFlowA * Power(fPopulation,fDesignFlowB) * 0.001 ) / EP;
  DFlowEp := Max(2*AFlowEp,dtmp);
end;

procedure TLCRConfig.GetFlows(Ownership, SourceWater, Population: integer;
                              var AFlow, DFlow: double);
var fAvgFlowA,fAvgFlowB,fDesignFlowA,fDesignFlowB,dtmp : double;
    fPopulation : double;
begin
  fAvgFlowA:=FlowVars[Ownership+1,SourceWater,1].V;
  fAvgFlowB:=FlowVars[Ownership+1,SourceWater,2].V;
  fDesignFlowA:=DFlowVars[Ownership+1,SourceWater,1].V;
  fDesignFlowB:=DFlowVars[Ownership+1,SourceWater,2].V;

  fPopulation := Population;

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

function TLCRConfig.GetOccRegion(State: string): integer;
var i : integer;
begin
  Result:=-1;
  i:=OccMap.IndexOf(uppercase(State));
  if i>=0 then
    Result:=Integer(OccMap.Objects[i]);
end;

procedure TLCRConfig.Load(FileName: string);
var
  I: TBiggerIniFile;
  P: pointer;
  S: TMemoryStream;
  ii, jj: integer;
  TP: TPicked;
  iSystemType: integer;
begin
  I := TBiggerIniFile.Create(FileName);

  RunName := I.ReadString('General','RunName',RunName);
  OptionName := I.ReadString('General','OptionName',OptionName);
  RunDescription := I.ReadString('General','RunDescription',RunDescription);
  PWSDataFile := I.ReadString('General','PWSDataFile',PWSDataFile);
  BasePWSDataFile := I.ReadString('General','BasePWSDataFile',BasePWSDataFile);
  ScenPWSDataFile := I.ReadString('General','ScenPWSDataFile',ScenPWSDataFile);
  EPProbsXL := I.ReadString('General','EPProbsFile',EPProbsXL);
  DiscRateList := I.ReadString('General','DiscRateList',DiscRateList);
  DiscountRate := I.ReadFloat('General','DiscountRate',DiscountRate);
  BaseCostSteps := I.ReadString('General','BaseCostSteps',BaseCostSteps);
  ScenCostSteps := I.ReadString('General','ScenCostSteps',ScenCostSteps);
  BaseVarData := I.ReadString('General','BaseVarData',BaseVarData);
  ScenVarData := I.ReadString('General','ScenVarData',ScenVarData);
  VariabilityRun := I.ReadBool('General','VariabilityRun',VariabilityRun);
  NumberOfVLoops := I.ReadInteger('General','NumberOfVLoops',NumberOfVLoops);
  fYearsOfAnalysis := I.ReadInteger('General','YearsOfAnalysis',fYearsOfAnalysis);
  fYearsOfOutput := I.ReadInteger('General','YearsOfOutput',fYearsOfOutput);
  RunOptionOnly := I.ReadBool('General','RunOptionOnly',RunOptionOnly);
  RunBaselineOnly := I.ReadBool('General','RunBaselineOnly',RunBaselineOnly);
  RunDifference := I.ReadBool('General','RunDifference',RunDifference);
  RunNoBLAveraging :=
I.ReadBool('General','RunNoBLAveraging',self.RunNoBLAveraging);
  NoICRCosts := I.ReadBool('General','NoICRCosts',NoICRCosts);
  CCTCostEquations :=I.ReadString('General','CCTCostEquations',CCTCostEquations);
  OutputLeadBins := I.ReadBool('General','OutputLeadBins',OutputLeadBins);
  Debug := I.ReadBool('General','Debug',Debug);
  UseCompiledCost := I.ReadBool('General','UseCompiledCost',UseCompiledCost);
  CCTToFull := I.ReadBool('General','CCTToFull',CCTToFull);
  SmallSystemFlexibility :=
I.ReadBool('General','SmallSystemFlexibility',SmallSystemFlexibility);
  CCTCostEquationLevel :=
I.ReadString('General','CCTCostEquationLevel',CCTCostEquationLevel);
  BenefitsAll := I.ReadBool('General','BenefitsAll',BenefitsAll);
  BenByYear := I.ReadBool('General','BenefitsByYear',BenByYear);

  IQValueSens:=I.ReadInteger('General','IQValueSens',IQValueSens);
  IQDRSens:=I.ReadInteger('General','IQDRSens',IQDRSens);
  CVDDRSens:=I.ReadInteger('General','CVDDRSens',CVDDRSens);
  ChildBLSens:=I.ReadInteger('General','ChildBLSens',ChildBLSens);
  VolLeadProg := I.ReadInteger('General','VolLeadProg',VolLeadProg);


  RevenueYear:=I.ReadInteger('Valuation','RevenueYear',RevenueYear);

RevenueInflationFactor:=I.ReadFloat('Valuation','RevenueInflationFactor',RevenueInfl
ationFactor);
  P:=@PWSRevenue;
  I.ReadBinaryData('PWSRevenue','Data',P,SizeOf(PWSRevenue));

   NumberOfTrials := I.ReadInteger('Uncertainty','NumberOfTrials',NumberOfTrials);
   MeanUncertLevel := I.ReadBool('Uncertainty','MeanUncertLevel',MeanUncertLevel);
   Seed := I.ReadInteger('Uncertainty','Seed',Seed);

   P:=@EntryPointProbs;
   I.ReadBinaryData('EntryPointProbs','Data',P,SizeOf(EntrypointProbs));
   ExpectedEP;

   S:=TMemoryStream.Create;
   for ii := 0 to OCategoryDefs.Count - 1 do
     TPicked(OCategorydefs.Items[ii]).Free;
   OCategoryDefs.Clear;
   I.ReadBinaryStream('Categories','Data',S);
   S.Position:=0;
   S.Read(ii,sizeof(ii));
   for jj:=1 to ii do begin
      TP:=TPicked.create;
      TP.LoadFromStream(S);
      OCategoryDefs.Add(TP);
   end;
   S.Free;

   iSystemType := 0;
   iSystemType := I.ReadInteger('General','SystemType',iSystemType);
   SystemType := TSystemType(iSystemType);
   SmallProxyPop := I.ReadInteger('General','SmallProxyPop',SmallProxyPop);
   PWS90PctBp1 := I.ReadInteger('General','PWS90PctBp1',PWS90PctBp1);
   PWS90PctBp2 := I.ReadInteger('General','PWS90PctBp2',PWS90PctBp2);
   SchoolOption := I.ReadString('General','SchoolOption',SchoolOption);
   RunSysType := I.ReadString('General','RunSysType',RunSysType);

   I.Free;
end;

procedure TLCRConfig.ModelPreCalcs;
begin
  ConnectExcelEPProbs;
end;

procedure TLCRConfig.PopulateUncertainty(Unc: TUncertaintyStudy);
begin

end;

procedure TLCRConfig.RenumberCategories;
var i,p : integer;
    n,s : string;
begin
  for i:=0 to OcategoryDefs.Count-1 do begin
```

```
    n:=TPicked(OcategoryDefs.Items[i]).Name;
    p:=pos(')',n);
    if p>0 then delete(n,1,p+1);

    if I+1<10 then  s:='  '+inttostr(I+1)+') ' else
    if i+1<100 then s:=' '+inttostr(i+1)+') ' else
                    s:=inttostr(I+1)+') ' ;

    TPicked(OcategoryDefs.Items[i]).Name:=s+n;
  end;
end;

procedure TLCRConfig.Save(FileName: string);
var I : TBiggerIniFile;
    P : pointer;
    S : TMemoryStream;
    tI : integer;
    iSystemType: integer;
begin
   I := TBiggerIniFile.Create(FileName);

   I.WriteString('General','RunName',RunName);
   I.WriteString('General','OptionName',OptionName);
   I.WriteString('General','RunDescription',RunDescription);
   I.WriteString('General','PWSDataFile',PWSDataFile);
   I.WriteString('General','BasePWSDataFile',BasePWSDataFile);
   I.WriteString('General','ScenPWSDataFile',ScenPWSDataFile);
   I.WriteString('General','EPProbsFile',EPProbsXL);
   I.WriteString('General','DiscRateList',DiscRateList);
   I.WriteFloat('General','DiscountRate',DiscountRate);
   I.WriteString('General','BaseCostSteps',BaseCostSteps);
   I.WriteString('General','ScenCostSteps',ScenCostSteps);
   I.WriteString('General','BaseVarData',BaseVarData);
   I.WriteString('General','ScenVarData',ScenVarData);
   I.WriteBool('General','VariabilityRun',VariabilityRun);
   I.WriteInteger('General','NumberOfVLoops',NumberOfVLoops);
   I.WriteInteger('General','YearsOfAnalysis',fYearsOfAnalysis);
   I.WriteInteger('General','YearsOfOutput',fYearsOfOutput);
   I.WriteBool('General','RunOptionOnly',RunOptionOnly);
   I.WriteBool('General','RunBaselineOnly',RunBaselineOnly);
   I.WriteBool('General','RunDifference',RunDifference);
   I.WriteBool('General','RunNoBLAveraging',RunNoBLAveraging);
   I.WriteBool('General','NoICRCosts',NoICRCosts);
   I.WriteString('General','CCTCostEquations',CCTCostEquations);
   I.WriteBool('General','OutputLeadBins',OutputLeadBins);
   I.WriteBool('General','Debug',Debug);
   I.WriteBool('General','UseCompiledCost',UseCompiledCost);
   I.WriteBool('General','CCTToFull',CCTToFull);
   I.WriteBool('General','SmallSystemFlexibility',SmallSystemFlexibility);
   I.WriteString('General','CCTCostEquationLevel',CCTCostEquationLevel);
   I.WriteBool('General','BenefitsAll',BenefitsAll);
   I.WriteBool('General','BenefitsByYear',BenByYear);

   I.WriteInteger('General','IQValueSens',IQValueSens);
   I.WriteInteger('General','IQDRSens',IQDRSens);
   I.WriteInteger('General','CVDDRSens',CVDDRSens);
   I.WriteInteger('General','ChildBLSens',ChildBLSens);
   I.WriteInteger('General','VolLeadProg',VolLeadProg);

   I.WriteInteger('Uncertainty','NumberOfTrials',NumberOfTrials);
   I.WriteBool('Uncertainty','MeanUncertLevel',MeanUncertLevel);
   I.WriteInteger('Uncertainty','Seed',Seed);
   I.WriteInteger('Valuation','RevenueYear',RevenueYear);
   I.WriteFloat('Valuation','RevenueInflationFactor',RevenueInflationFactor);
   P:=@PWSRevenue;
   I.WriteBinaryData('PWSRevenue','Data',P,SizeOf(PWSRevenue));

   P:=@EntryPointProbs;
   I.WriteBinaryData('EntryPointProbs','Data',P,SizeOf(EntrypointProbs));


   S:=TMemoryStream.Create;
   ti:=OcategoryDefs.Count;
   S.Write(ti,sizeof(ti));
   for ti:=0 to OcategoryDefs.Count-1 do begin
      TPicked(OCategoryDefs[ti]).SaveToStream(S);
   end;
   S.Position:=0;
   I.WriteBinaryStream('Categories','Data',S);
   S.Free;

   iSystemType := Integer(SystemType);
   I.WriteInteger('General','SystemType',iSystemType);
   I.WriteInteger('General','SmallProxyPop',SmallProxyPop);
   I.WriteInteger('General','PWS90PctBp1',PWS90PctBp1);
   I.WriteInteger('General','PWS90PctBp2',PWS90PctBp2);
   I.WriteString('General','SchoolOption',SchoolOption);
   I.WriteString('General','RunSysType',RunSysType);

   I.Free;
end;

procedure TLCRConfig.SetDR(value: string);
var T : TStringList;
    i : integer;
begin
```

```
   T:=TStringLIst.Create;
   try
     T.CommaText:=value;
     setlength(DiscRates,T.Count);
     for i:=0 to T.Count-1 do begin
        DiscRates[i]:=strtofloat(T.Strings[i]);
     end;
     fDiscStr:=value;
   except
     //TODO handle this
   end;
   T.Free;
end;

{ TRunLog }

procedure TRunLog.AddExtra(Tag, HTML: string);
begin
  fExtras.Add(Tag+'!'+HTML);
end;

constructor TRunLog.Create;
begin
  inherited;
  fLog:=TStringList.Create;
  fExtras:=TstringList.Create;
end;

destructor TRunLog.Destroy;
begin
  fLog.Free;
  fExtras.Free;
  inherited;
end;


function TRunLog.GetExtra(Tag : string): string;
var i,p : integer;
    t : string;
begin
  Result:='Unknown';
  for i:=0 to fExtras.Count-1 do begin
    p:=pos('!',fExtras[i]);
    t:=copy(fExtras[i],1,p-1);
    if CompareText(t,Tag)=0 then begin
      Result:=copy(fExtras[i],P+1,Length(fExtras[i]));
      break;
    end;
  end;
end;

function TRunLog.HTMLLog: string;
var i : integer;
begin
  for i:=0 to fLog.Count-1 do begin
     Result:=Result+fLog[i]+'<br>';
  end;
end;

procedure TRunLog.LoadFromStream(Strm: TStream);

begin
  fLog.Text:=ReadStreamStr(Strm);
  fExtras.Text:=ReadStreamStr(Strm);
end;

procedure TRunLog.put_log_txt(txt: string);
var T : string;
begin
   T:=FormatDateTime('mm-dd-yy hh:mm:ss',Now);
   fLog.Add(T+'  '+Txt);
end;

procedure TRunLog.put_log_txt_strings(txt:TstringList);
var i : integer;
begin
  for i:=0 to txt.Count-1 do
    fLog.Add(Txt[i]);
end;
procedure TRunLog.put_log_txt_nostamp(txt: string);
begin
   fLog.Add(Txt);
end;

procedure TRunLog.SaveToStream(Strm: TStream);
begin
  WriteStreamStr(Strm,fLog.Text);
  WriteStreamStr(Strm,fExtras.Text);
end;

procedure TRunLog.WriteIt(aFile: string);
var t : TextFile;
    i : integer;
begin
  assignfile(T,aFile);
  rewrite(T);
  for i:=0 to fLog.Count-1 do begin
     writeln(t,fLog[i]);
  end;
  closefile(T);
end;

end.
```