```
unit LCRCostVars;

interface

uses SysUtils, StrUtils, Math, Generics.Collections, Classes,
     System.IOUtils,DB,SafewaterUncertBucket,LCRGlobals, LCRConfig;

type
  TRandomGiver=class
    fBase,fScen : array[1..1000000] of double;
    fBIX, fSIX,fSC,fBC : integer;
    procedure MakeSaveOpen;
    function Open : boolean;
    function GetS : double;
    function GetB : double;
  end;

  TCostVarParams = record
    p1,p2,p3 : double;
    CustomPoints : array of double;
  end;

  //integer = mash of dimensions
  TCostVarParamsData = TDictionary<integer,TCostVarParams>;
  TCostVars=class;

  TCostVar=class
    fID,fIDR : string;
    fBySize,fBySource,fByLSL,fByCCT,fByType : boolean;
    fData : TCostVarParamsData;
    fDistType : integer;
    fImAProb, fSameBaseline, fChangesYearly, fImSpecial: boolean;
    fIDX : integer;
    fBaselineVar,fNext : TCostVar;
    fCurValue, fRawValue : double;
    fCostVarParams: TCostVarParams;
    fSeedValue, fUseSeed : integer;
    fSeedSet : boolean;

    constructor CreateFromDS(d : TDataset; BaselineVars : TCostVars=nil);
    constructor CreateZero(aName : string);
    procedure AddFromDS(d : TDataset);
    destructor Destroy; override;

    function MakeId(const aSz,aSrc,aLSL,aCCT,aType : integer) : integer;
    procedure GetRandomValue(const aSz, aSrc, aLSL, aCCT, aType : integer; const
SetProbsTo01 : boolean;
                             var CurValue, RawValue: double; IsBaseline: boolean);
    function GetMeanValue(const aSz, aSrc, aLSL, aCCT, aType : integer; const
SetProbsTo01 : boolean): double;
    procedure GetCurValue(var CurValue, RawValue: double);
    procedure GetCostVarParams(const aSz, aSrc, aLSL, aCCT, aType : integer);
    procedure GetRandomValue2(const aSz, aSrc, aLSL, aCCT, aType : integer; const
SetProbsTo01 : boolean; var CurValue, RawValue: double;
                              const BaselineCols, BaselineData: TStringList);
    procedure GetRandomValueEP(const aSz, aSrc, aLSL, aCCT, aType, nEps: integer;
const SetProbsTo01 : boolean;
                               var CurValue, RawValue: double; IsBaseline: boolean);

    procedure SetCustomSeed(PWSSeed : integer);
  private
    function GetStrVal(s: string): integer;
  end;

  TPWS_p_valuesRec = record
    p_lsl: integer;

    p_inventory: integer;
    p_tap_nine: integer;
    p_tap_annual: integer;
    p_tap_triennial: integer;

    p_spec_req: integer;

    p_wqp_annual: integer;
    p_wqp_triennial: integer;
    p_wqp_six_red: integer;

    p_b3: integer;

    pbaseph: integer;
    pbasepo4: integer;
    pbasephpo4: integer;
    baselinepo4dose: integer;
    baselineph_wph: integer;
    baselineph_woph: integer;
    baselineph_wocct: integer;
    baselineph_wpo4ph: integer;

    perc_lsl: double;

    ppBin1: double;
    ppBin2: double;
    ppBin3: double;
    ppAdjBin1: double;
    ppAdjBin2: double;
    ppAdjBin3: double;
  end;

  TCostVars=class(TObjectDictionary<string,TCostVar>)
    NumVars : integer;
    SpecialVars : TStringList;
    DirectArray : TArray<TCostVar>;

    constructor Create(D,V : TDataset; DataSpreadsheet : string; BaselineVars : 
TCostVars=nil);
    destructor Destroy; override;

    procedure ResetRandomSeeds(PWSSeed : integer);

    procedure FillValueArray(var a1, a2 : TDoubleArray; const aSz, aSrc, aLSL, aCCT, 
aType, Yr, Bp1, Bp2: integer;
                             const SetProbsTo01, ForceDraw: boolean;
                             O: TDictionary<string,double>; IsBaseline: boolean);
    procedure ReadVars(Filename: string);
    function Calculate_p_lsl(const aSz, aSrc, aLSL, aCCT, aType : integer; const 
SetProbsTo01: boolean;
                             GetBaseCurValue : boolean=false): double;
    procedure Calculate_pws_p_values(const aSz, aSrc, aLSL, aCCT, aType : integer; 
const SetProbsTo01 : boolean;
                                     var p_values: TPWS_p_valuesRec; GetBaseCurValue 
: boolean=false);
    procedure FillValueArray2(var a1, a2 : TDoubleArray; const aSz, aSrc, aLSL, 
aCCT, aType, Yr, Bp1, Bp2: integer;
                              const SetProbsTo01, ForceDraw : boolean;
                              const BaselineCols, BaselineData: TStringList);
    procedure DrawLSLReplacementRates(const aSz, aSrc, aLSL, aCCT, aType, aYrs : 
integer; const option: string;
                                      var a1: TDoubleArray);
    function Calculate_bin_distr(const aSz, aSrc, aLSL, aCCT, aType : integer; const 
SetProbsTo01: boolean;
                                 GetBaseCurValue : boolean=false): double;
    function Calculate_dist_lead_base(const iBin, bp1, bp2, aSz, aSrc, aLSL, aCCT, 
aType : integer;
                                      const SetProbsTo01: boolean; GetBaseCurValue : 
boolean=false): double;
    procedure DrawP_Source_Chng(const aSz, aSrc, aLSL, aCCT, aType, aYrs, nEps: 
integer; const option: string;
                                var a1: TDoubleArray);
    procedure DrawP_Source_Sig(const aSz, aSrc, aLSL, aCCT, aType, aYrs, nEps: 
integer; const option: string;
                                var a1: TDoubleArray);
    procedure DrawP_Treat_Change(const aSz, aSrc, aLSL, aCCT, aType, aYrs, nEps: 
integer; const option: string;
```

```
                                 var a1: TDoubleArray);
    procedure DrawPp_LSL_Replaced_Vol_Pct(const aSz, aSrc, aLSL, aCCT, aType, aYrs:
integer; const option: string;
                                          var a1: TDoubleArray);
    function Calculate_Num_tap_ge_al(const numb, prob: double): double;
  end;

  function iiRandom(IsBaseline: boolean): double;
  function iiRandomRange(const a,b : integer; IsBaseline : boolean) : integer;

var
   NoRandom : boolean;
   RG : TRandomGiver;
   UserSeeds : boolean;

implementation

uses VCL.FlexCel.Core, FlexCel.XlsAdapter;

function iRandom: double;
begin
  if NoRandom then
    Result:=0.3
  else
    Result:=Random;
end;

function iRandomRange(const a,b : integer) : integer;
begin
  if NoRandom then
    Result:=a
  else
    Result:=RandomRange(a,b);
end;

function iiRandom(IsBaseline: boolean): double;
begin
  if NoRandom then
    Result:=0.3
  else
  begin
    if IsBaseline then Result:=RG.GetB else Result:=RG.GetS;
  end;
end;

function iiRandomRange(const a,b : integer; IsBaseline : boolean) : integer;
var r : double;
begin
```

```
  if NoRandom then
    Result:=a
  else begin
    if IsBaseline then r:=RG.GetB else R:=RG.GetS;
    Result:=Round(a + (b-a)*R);
  end;
end;


{ TRandomGiver }

function TRandomGiver.GetB: double;
begin
  if UserSeeds then begin
    Result:=Random;
    inc(fBC);
    exit;
  end;
  Result:=fBase[fBIX];
  inc(fBix);
  if fBix>high(fBase) then fBix:=low(fBase);
end;

function TRandomGiver.GetS: double;
begin
  if UserSeeds then begin
    Result:=Random;
    inc(fSC);
    exit;
  end;
  Result:=fScen[fSIX];
  inc(fSix);
  if fSix>high(fScen) then fSix:=low(fScen);
end;

procedure TRandomGiver.MakeSaveOpen;
var i : integer;
    T : TBufferedFileStream;
begin
   for i:=Low(fBase) to high(fBase) do begin
     fBase[i]:=Random;
     fScen[i]:=Random;
   end;
   T:=TBufferedFileStream.Create('b2.rnd',fmCreate,4096);
   T.WriteBuffer(fBase[1],sizeof(fBase));
   T.Free;
   T:=TBufferedFileStream.Create('s2.rnd',fmCreate,4096);
   T.WriteBuffer(fScen[1],sizeof(fScen));
   T.Free;
   Open;
end;

function TRandomGiver.Open: boolean;
var T : TBufferedFileStream;
begin
  if TFile.Exists('b2.rnd') then begin
     T:=TBufferedFileStream.Create('b2.rnd',fmOPenRead,4096);
     T.ReadBuffer(fBase[1],SizeOf(fBase));
     T.Free;
     T:=TBufferedFileStream.Create('s2.rnd',fmOPenRead,4096);
     T.ReadBuffer(fScen[1],SizeOf(fScen));
     T.Free;
     fBIX:=1;
     fSIX:=1;
     fSC:=0;
     fBC:=0;
  end else
    MakeSaveOpen;
end;

{ TCostVar }

procedure TCostVar.AddFromDS(d: TDataset);
var id,i : integer;
    P : TCostVarParams;
    CV : TstringList;
begin
  //from InputValues database

id:=MakeID(D.FieldByName('SystemSize').AsInteger,D.FieldByName('SourceWater').AsInte
ger,
              D.FieldByName('LSL').AsInteger,D.FieldByName('CCT').AsInteger,
              D.FieldByName('SystemType').AsInteger);
   setlength(P.CustomPoints,0);
   p.p1:=0; p.p2:=0; p.p3:=0;
   case fDistType of
      dNone : P.p1:=d.FieldByName('XValue').AsFloat;
      dUniform : begin
                   P.p1:=d.FieldByName('MinValue').AsFloat;
                   P.p2:=d.FieldByName('MaxValue').AsFloat;
                 end;
      dBeta    : begin
                   P.p1:=d.FieldByName('MinValue').AsFloat;    // Alpha
                   P.p2:=d.FieldByName('MaxValue').AsFloat;    // Beta
                 end;
      dTriangular : begin
```

```
                     P.p1:=d.FieldByName('MinValue').AsFloat;
                     P.p2:=d.FieldByName('MaxValue').AsFloat;
                     P.p3:=d.FieldByName('MostLikely').AsFloat;
                    end;
      dICustom :begin
                  CV:=TstringList.Create;
                  CV.CommaText:=d.FieldByName('Custom').AsString;
                  setlength(P.CustomPoints,CV.Count);
                  for i:=0 to CV.Count-1 do
                    P.CustomPoints[i]:=strtofloat(CV[i]);
                  CV.Free;
                end
   else
//     raise exception.Create('Need to implement dist:'+inttostr(fDistType));
   end;
   fData.Add(id,P);
end;

function GetDistMean(dtype:integer;parm1,parm2,parm3:double):double;
begin
   case Dtype of
     dNormal      : Result:=parm1;
     dTriangular  : Result:=Parm3;
     dUniform     : Result:=(parm1+Parm2) / 2;
     dICustom     : raise exception.Create('Need to handle ICustom Mean in
GetDistMean'); //Result:=???;
     dBeta        : raise exception.Create('Need to handle Beta Mean in
GetDistMean'); //Result:=???;
     dnone        : Result:=parm1;
   else
     raise exception.Create('Unknown dist type in GetDistMean');
   end; {case}
end;

function  StrDistToCode(N : string) : integer;
var S : string;
begin
  S:=UpperCase(N);
  //intercept "Cutom" from database and call it ICustom
  if s='CUSTOM' then result:=dICustom else
  if s='NORMAL' then result:=dNormal else
  if s='TRIANGULAR' then result:=dTriangular else
  if s='POISSON' then result:=dPoisson else
  if s='BINOMIAL' then result:=dBinomial else
  if s='LOGNORMAL' then result:=dLogNormal else
  if s='UNIFORM' then result:=dUniform else
  if s='EXPONENTIAL' then result:=dExponential else
  if s='GEOMETRIC' then result:=dGeometric else
  if s='WEIBULL' then result:=dWeibull else
  if s='GAMMA' then result:=dGamma else
  if s='LOGISTIC' then result:=dLogistic else
  if s='CAUCHY' then result:=dCauchy else
  if s='PARETO' then result:=dPareto else
  if s='BETA' then result:=dBeta else
  if s='H-M-L' then result:=dHML else
  if s='VARIABLECUSTOM' then result:=dVariableCustom else
  if s='ICUSTOM' then result:=dICustom else
     result:=dNone;
end;

function TCostVar.GetStrVal(s : string): integer;
var i : integer;
begin
  Result:=0;
  for i:=1 to length(s) do
    Result:=Result+Ord(s[i]);
end;

constructor TCostVar.CreateFromDS(d: TDataset; BaselineVars : TCostVars=nil);
begin
  inherited create;
  //from InputDesc database
  fData:=TCostVarParamsData.Create;
  fID:=D.FieldByName('ID_Name').AsString;
  fIDR:=fID+'_r';
  fBySize:=D.FieldByName('SystemSizeDep').AsString='Y';
  fBySource:=D.FieldByName('SourceWaterDep').AsString='Y';
  fByLSL:=D.FieldByName('LSLDep').AsString='Y';
  fByCCT:=D.FieldByName('CCTDep').AsString='Y';
  fByType:=D.FieldByName('SystemTypeDep').AsString='Y';
  fDistType:=StrDistToCode(D.FieldByName('Distribution').AsString);
  fImAProb:=pos('P_',uppercase(fID))=1;
  fImSpecial:=False;
  fNext:=nil;
  fSeedValue:=GetStrVal(lowercase(fID));

  if (fID = 'pp_lsl_replaced_one') or
     (fID = 'pp_lsl_replaced_two') or
     (fID = 'pp_lsl_replaced_three')  then
    fChangesYearly := True
  else
    fChangesYearly := False;

  if Assigned(BaselineVars) then begin
    fSameBaseline:=D.FieldByName('BaselineSame').AsString='Y';
    BaselineVars.TryGetValue(fID, fBaselineVar);
  end else
    fSameBaseline:=False;
end;

constructor TCostVar.CreateZero(aName : string);
var P : TCostVarParams;
begin
  inherited create;
  //from InputDesc database
  fData:=TCostVarParamsData.Create;
  fID:=aName;
  fSeedValue:=GetStrVal(lowercase(fID));
  fIDR:=fID+'_r';
  fBySize:=False;
  fBySource:=False;
  fByLSL:=False;
  fByCCT:=False;
  fByType:=False;
  fDistType:=dNone;
  fImAProb:=False;
  fSameBaseline:=False;
  fChangesYearly:=False;
  fNext:=nil;
  fImSpecial:=false;
  setlength(P.CustomPoints,0);
  p.p1:=0; p.p2:=0; p.p3:=0;
  fData.Add(0,P);
end;

destructor TCostVar.Destroy;
begin
  fData.Free;
  inherited;
end;

procedure TCostVar.GetRandomValue(const aSz, aSrc, aLSL, aCCT, aType: integer; const
SetProbsTo01: boolean;
                                  var CurValue, RawValue: double; IsBaseline:
boolean);
var id,i : integer;
    V : TCostVarParams;
    r,tmp : double;
begin
  fCurValue:=0; fRawValue:=0;
  if not fSeedSet then begin
    RandSeed:=fUseSeed;
    fSeedSet:=true;
  end;
  if fSameBaseline then begin
    fBaselineVar.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01,
CurValue, RawValue, IsBaseline);
    fCurValue:=CurValue; fRawValue:=RawValue;
  end else begin
    id:=MakeID(aSz, aSrc, aLSL, aCCT, aType);
    R:=iiRandom(IsBaseline);
    fCurValue:=0; fRawValue:=0;
    if fData.TryGetValue(id,V) then begin
      if fDistType=dICustom then begin
        tmp:=0;
        i:=1;
        repeat
          tmp:=tmp+V.CustomPoints[i-1];
          fCurValue:=i;
          inc(i);
          if i>length(V.CustomPoints) then break;
        until tmp>=R;
      end else
        fCurValue:=icdf(fDistType,r,v.p1,v.p2,v.p3);

      fRawValue:=fCurValue;
      if (fImAProb) and (SetProbsTo01) then begin
        if R<=fCurValue then fCurValue:=1 else fCurValue:=0;
      end;
    end else begin
      //?? raise exception probably
    end;
  end;
  CurValue:=fCurValue; RawValue:=fRawValue;
end;

procedure TCostVar.GetRandomValue2(const aSz, aSrc, aLSL, aCCT, aType : integer;
  const SetProbsTo01: boolean; var CurValue, RawValue: double;
  const BaselineCols, BaselineData: TStringList);
var id,i : integer;
    V : TCostVarParams;
    r,tmp : double;

    optParamP1, optParamP2, optParamP3: double;
    blParamP1, blParamP2, blParamP3: double;
    fSameStrata: boolean;
    name: string;
begin
  fCurValue:=0; fRawValue:=0;
  fSameStrata := false;
  name := fID;
  if not fSeedSet then begin
    RandSeed:=fUseSeed;
    fSeedSet:=true;
  end;

  GetCostVarParams(aSz, aSrc, aLSL, aCCT, aType);
  optParamP1 := fCostVarParams.p1;
  optParamP2 := fCostVarParams.p2;
  optParamP3 := fCostVarParams.p3;

  if Assigned(fBaselineVar) then
  begin
    fBaselineVar.GetCostVarParams(aSz, aSrc, aLSL, aCCT, aType);
    blParamP1 := fBaselineVar.fCostVarParams.p1;
    blParamP2 := fBaselineVar.fCostVarParams.p2;
    blParamP3 := fBaselineVar.fCostVarParams.p3;

    if (optParamP1 > 0) and
       (optParamP1 = blParamP1) and
       (optParamP2 = blParamP2) and
       (optParamP3 = blParamP3) then
      fSameStrata := true;
  end;

  if fSameStrata then
  begin
    fCurValue := BaselineData.Strings[BaselineCols.IndexOf(fID)].ToDouble;
    if fImAProb then
      fRawValue := BaselineData.Strings[BaselineCols.IndexOf(fID+'_r')].ToDouble;
  end
  else
  if fSameBaseline then begin
    //fBaselineVar.GetRandomValue(aSz, aSrc, aLSL, aCCT, SetProbsTo01, CurValue,
RawValue);
    fCurValue := BaselineData.Strings[BaselineCols.IndexOf(fID)].ToDouble;
    if fImAProb then
      fRawValue := BaselineData.Strings[BaselineCols.IndexOf(fID+'_r')].ToDouble;
  end else begin
    id:=MakeID(aSz, aSrc, aLSL, aCCT, aType);
    R:=iiRandom(false); // changed to iiRandom from iRandom on 4/5/18
    fCurValue:=0; fRawValue:=0;
    if fData.TryGetValue(id,V) then begin
      if fDistType=dICustom then begin
        tmp:=0;
        i:=1;
        repeat
          tmp:=tmp+V.CustomPoints[i-1];
          fCurValue:=i;
          inc(i);
          if i>length(V.CustomPoints) then break;
        until tmp>=R;
      end else
        fCurValue:=icdf(fDistType,r,v.p1,v.p2,v.p3);

      fRawValue:=fCurValue;
      if (fImAProb) and (SetProbsTo01) then begin
        if R<=fCurValue then fCurValue:=1 else fCurValue:=0;
      end;
    end else begin
      //?? raise exception probably
    end;
  end;
  CurValue:=fCurValue; RawValue:=fRawValue;

end;

procedure TCostVar.GetRandomValueEP(const aSz, aSrc, aLSL, aCCT, aType, nEps: 
integer;
  const SetProbsTo01: boolean; var CurValue, RawValue: double;
  IsBaseline: boolean);
var id,i : integer;
    V : TCostVarParams;
    r,tmp : double;
begin
  fCurValue:=0; fRawValue:=0;
  if fSameBaseline then begin
    fBaselineVar.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, 
CurValue, RawValue, IsBaseline);
    fCurValue:=CurValue; fRawValue:=RawValue;
  end else begin
    id:=MakeID(aSz, aSrc, aLSL, aCCT, aType);
    R:=iiRandom(IsBaseline);
    fCurValue:=0; fRawValue:=0;
    if fData.TryGetValue(id,V) then begin
      if fDistType=dICustom then begin
        tmp:=0;
        i:=1;
        repeat
          tmp:=tmp+V.CustomPoints[i-1];
          fCurValue:=i;
          inc(i);
          if i>length(V.CustomPoints) then break;
        until tmp>=R;
      end else
      begin
        v.p1 := v.p1 / nEps;
        fCurValue:=icdf(fDistType,r,v.p1,v.p2,v.p3);
      end;

      fRawValue:=fCurValue;
      if (fImAProb) and (SetProbsTo01) then begin
        if R<=fCurValue then fCurValue:=1 else fCurValue:=0;
      end;
    end else begin
      //?? raise exception probably
    end;
  end;
  CurValue:=fCurValue; RawValue:=fRawValue;
end;

procedure TCostVar.GetCostVarParams(const aSz, aSrc, aLSL, aCCT, aType: integer);
var id : integer;
    V : TCostVarParams;
begin
  id:=MakeID(aSz, aSrc, aLSL, aCCT, aType);
  if fData.TryGetValue(id,V) then begin
    fCostVarParams.p1 := v.p1;
    fCostVarParams.p2 := v.p2;
    fCostVarParams.p3 := v.p3;
  end
  else
  begin
    fCostVarParams.p1 := 0;
    fCostVarParams.p2 := 0;
    fCostVarParams.p3 := 0;
  end;
end;

procedure TCostVar.GetCurValue(var CurValue, RawValue: double);
begin
  CurValue:=fCurValue;
  RawValue:=fRawValue;
end;

function TCostVar.GetMeanValue(const aSz, aSrc, aLSL, aCCT, aType : integer; const
SetProbsTo01 : boolean): double;
var id : integer;
    V : TCostVarParams;
    r : double;
begin
  fCurValue:=0;
  if fSameBaseline then begin
    fCurValue:=fBaselineVar.GetMeanValue(aSz, aSrc, aLSL, aCCT, aType,
SetProbsTo01);
```

```
  end else begin
    id:=MakeID(aSz, aSrc, aLSL, aCCT, aType);
    R:=Random;
    if fData.TryGetValue(id,V) then begin
      fCurValue:=GetDistMean(fDistType,v.p1,v.p2,v.p3);
      if (fImAProb) and (SetProbsTo01) then begin
        if R<=fCurValue then fCurValue:=1 else fCurValue:=0;
      end;
    end else begin
      //?? raise exception probably
    end;
  end;
  Result:=fCurValue;
end;


function TCostVar.MakeId(const aSz, aSrc, aLSL, aCCT,aType : integer): integer;
begin
  Result:=0;
  if fByType then Result:=Result+aType*100000000;
  if fBySize then Result:=Result+aSz*1000000;
  if fBySource then Result:=Result+aSRc*10000;
  if fByLSL then Result:=Result+aLSL*100;
  if fByCCT then Result:=Result+aCCT;
end;

procedure TCostVar.SetCustomSeed(PWSSeed: integer);
begin
  //this change assumes PWS will continue to be handled in isolation....
  fUseSeed:=fSeedValue+PWSSeed;
  fSeedSet:=False;
end;

{ TCostVars }

function TCostVars.Calculate_bin_distr(const aSz, aSrc, aLSL, aCCT, aType : integer;
  const SetProbsTo01: boolean; GetBaseCurValue: boolean): double;
var v: TCostVar;
    curval, rawval: double;
begin
  Result := 0;
  v:=nil;
  TryGetValue('bin_distr',v);
  if not Assigned(v) then raise exception.Create('Cannot find bin_distr in
Calculate_bin_distr');

  if (v.fSameBaseline) and (GetBaseCurValue) then begin
    v.fBaselineVar.GetCurValue(curval, rawval);
    Result := curval;
  end else begin
    v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, rawval,
True);
    Result := curval;
  end;
end;

function TCostVars.Calculate_dist_lead_base(const iBin, bp1, bp2, aSz, aSrc, aLSL,
  aCCT, aType : integer; const SetProbsTo01: boolean; GetBaseCurValue: boolean):
double;
var v: TCostVar;
    curval, rawval: double;
    dbvar: string;
    fnd: boolean;
begin
  Result := 0;
  fnd := false;

  case iBin of
    1: dbvar := 'dist_lead_base_bin1';
    2: dbvar := 'dist_lead_base_bin2';
    3: dbvar := 'dist_lead_base_bin3';
  end;

  v:=nil;
  TryGetValue(dbvar,v);
  if not Assigned(v) then raise exception.Create('Cannot find '+dbvar +' in
Calculate_dist_lead_base');

  if (v.fSameBaseline) and (GetBaseCurValue) then begin
    v.fBaselineVar.GetCurValue(curval, rawval);
    Result := curval;
  end else begin
    while not fnd do begin
      v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, rawval,
True);
      curval := curval * 1000;
      case iBin of
        3: begin if curval <= bp1 then fnd := true; end;
        2: begin if (curval > bp1) and (curval <= bp2) then fnd := true; end;
        1: begin if curval > bp2 then fnd := true; end;
      end;
    end;
    Result := curval;
  end;
end;
```

```
function TCostVars.Calculate_Num_tap_ge_al(const numb, prob: double): double;
var r: double;
begin
  r := iiRandom(false);
  Result := icdf(dBinomial, r, prob, numb, 0);
end;

procedure TCostVars.Calculate_pws_p_values(const aSz, aSrc, aLSL, aCCT, aType :
integer;
  const SetProbsTo01: boolean; var p_values: TPWS_p_valuesRec; GetBaseCurValue :
boolean=false);
var v: TCostVar;
    curval, rawval: double;
    i: integer;
    tap: array[1..3] of double;
    wqp: array[1..3] of double;
    cct_trtmnt: array[1..3] of double;
    r, tp_tap, tp_wqp, tp_cct_trtmnt: double;

    procedure SetVal;
    begin
      if (v.fSameBaseline) and (GetBaseCurValue) then begin
        v.fBaselineVar.GetCurValue(curval, rawval);
      end else begin
        v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, rawval,
True);
      end;
    end;

begin
  for i := 1 to 3 do
  begin
    tap[i] := 0;
    wqp[i] := 0;
    cct_trtmnt[i] := 0;
  end;

  for v in Values do begin
    if v.fID = 'p_inventory' then
    begin
      SetVal;
      p_values.p_inventory := Round(curval);
    end
    else
    if v.fID = 'p_tap_nine' then
    begin
      SetVal;
      tap[1] := rawval;
```

```
end
else
if v.fID = 'p_tap_annual' then
begin
  SetVal;
  tap[2] := rawval;
end
else
if v.fID = 'p_tap_triennial' then
begin
  SetVal;
  tap[3] := rawval;
end
else
if v.fID = 'p_spec_req' then
begin
  SetVal;
  p_values.p_spec_req := Round(curval);
end
else
if v.fID = 'p_wqp_annual' then
begin
  SetVal;
  wqp[1] := rawval;
end
else
if v.fID = 'p_wqp_triennial' then
begin
  SetVal;
  wqp[2] := rawval;
end
else
if v.fID = 'p_wqp_six_red' then
begin
  SetVal;
  wqp[3] := rawval;
end
else
if v.fID = 'p_b3' then
begin
  SetVal;
  p_values.p_b3 := Round(curval);
end
else
if v.fID = 'perc_lsl' then
begin
  SetVal;
  p_values.perc_lsl := curval;
```

```
end
else
if v.fID = 'pbaseph' then
begin
  SetVal;
  cct_trtmnt[1] := rawval;
end
else
if v.fID = 'pbasepo4' then
begin
  SetVal;
  cct_trtmnt[2] := rawval;
end
else
if v.fID = 'pbasephpo4' then
begin
  SetVal;
  cct_trtmnt[3] := rawval;
end
else
if v.fID = 'baselinepo4dose' then
begin
  SetVal;
  p_values.baselinepo4dose := Round(curval);
end
else
if v.fID = 'baselineph_wph' then
begin
  SetVal;
  p_values.baselineph_wph := Round(curval);
end
else
if v.fID = 'baselineph_woph' then
begin
  SetVal;
  p_values.baselineph_woph := Round(curval);
end
else
if v.fID = 'baselineph_wocct' then
begin
  SetVal;
  p_values.baselineph_wocct := Round(curval);
end
else
if v.fID = 'baselineph_wpo4ph' then
begin
  SetVal;
  p_values.baselineph_wpo4ph := Round(curval);
  end
  else
  if v.fID = 'pbin1' then
  begin
    SetVal;
    p_values.ppBin1 := rawval;
  end
  else
  if v.fID = 'pbin2' then
  begin
    SetVal;
    p_values.ppBin2 := rawval;
  end
  else
  if v.fID = 'pbin3' then
  begin
    SetVal;
    p_values.ppBin3 := rawval;
  end
  else
  if v.fID = 'padjbin1' then
  begin
    SetVal;
    p_values.ppAdjBin1 := rawval;
  end
  else
  if v.fID = 'padjbin2' then
  begin
    SetVal;
    p_values.ppAdjBin2 := rawval;
  end
  else
  if v.fID = 'padjbin3' then
  begin
    SetVal;
    p_values.ppAdjBin3 := rawval;
  end;
end;

tp_tap := 0;
tp_wqp := 0;
for i := 1 to 3 do
begin
  tp_tap := tp_tap + tap[i];
  tp_wqp := tp_wqp + wqp[i];
end;

for i := 1 to 3 do
```

```
begin
  if tp_tap>0 then
    tap[i] := tap[i]/tp_tap;
  if tp_wqp>0 then
    wqp[i] := wqp[i]/tp_wqp;
end;

p_values.p_tap_nine := 0;
p_values.p_tap_annual := 0;
p_values.p_tap_triennial := 0;
if tp_tap>0 then begin
  tp_tap := 0;
  r := iirandom(true);
  for i := 1 to 3 do
  begin
    tp_tap := tp_tap + tap[i];
    if r<tp_tap then begin
      case i of
        1: p_values.p_tap_nine := 1;
        2: p_values.p_tap_annual := 1;
        3: p_values.p_tap_triennial := 1;
      end;
      break;
    end;
  end;
end;

p_values.p_wqp_annual := 0;
p_values.p_wqp_triennial := 0;
p_values.p_wqp_six_red := 0;
if tp_wqp>0 then begin
  tp_wqp := 0;
  r := iirandom(true);
  for i := 1 to 3 do
  begin
    tp_wqp := tp_wqp + wqp[i];
    if r<tp_wqp then begin
      case i of
        1: p_values.p_wqp_annual := 1;
        2: p_values.p_wqp_triennial := 1;
        3: p_values.p_wqp_six_red := 1;
      end;
      break;
    end;
  end;
end;

p_values.pbaseph := 0;
```

```
  p_values.pbasepo4 := 0;
  p_values.pbasephpo4 := 0;

  tp_cct_trtmnt := 0;
  // changed from irandom to iirandom(true) 4/6/18
  r := iirandom(true);
  for i := 1 to 3 do
  begin
    tp_cct_trtmnt := tp_cct_trtmnt + cct_trtmnt[i];
    if r<tp_cct_trtmnt then begin
      case i of
        1: p_values.pbaseph := 1;
        2: p_values.pbasepo4 := 1;
        3: p_values.pbasephpo4 := 1;
      end;
      break;
    end;
  end;
end;

function TCostVars.Calculate_p_lsl(const aSz, aSrc, aLSL, aCCT, aType : integer;
  const SetProbsTo01: boolean; GetBaseCurValue : boolean=false): double;
var v: TCostVar;
    curval, rawval: double;
begin
  Result := 0;
  for v in Values do begin
    if v.fID = 'p_lsl' then
    if (v.fSameBaseline) and (GetBaseCurValue) then begin
      v.fBaselineVar.GetCurValue(curval, rawval);
      Result := curval;
      break;
    end else begin
      v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, rawval,
True);
      Result := curval;
      break;
    end;
  end;
end;

constructor TCostVars.Create(D,V: TDataset; DataSpreadsheet : string; BaselineVars :
TCostVars=nil);
var T : TCostVar;
    i : integer;

begin
  inherited Create([doOwnsValues]);
  SpecialVars:=TstringList.Create;
  SpecialVars.Sorted:=True;

SpecialVars.CommaText:='p_sal_one,p_sal_two,p_sal_three,p_source_chng,p_treat_change
,'+
  'p_wqp_chng,p_lead_agg,p_copper_ale,p_lead_ale,p_source_chng,p_treat_change';

  D.First;
  NUmVars:=0;
  while not D.Eof do begin
    T:=TCostVar.CreateFromDS(D,BaselineVars);
    Add(T.fID,T);
    inc(NumVars);
    D.Next;
  end;
  V.First;
  while not V.Eof do begin
    if TryGetValue(V.FieldByName('ID_Name').AsString,T) then begin
      T.AddFromDS(V);
    end else begin
      //?? raise exception probably..
    end;
    V.Next;
  end;

  ReadVars(DataSpreadsheet);

  i:=0;
  for T in Values do begin
    T.fIDX:=i;
    if SpecialVars.IndexOf(T.fID)>=0 then
      T.fImSpecial:=True;
    inc(i);
  end;

  //Set up mutually exclusive relationships....
  if TryGetValue('p_tap_nine',T) then begin
    Items['p_tap_nine'].fNext:=Items['p_tap_annual'];
    Items['p_tap_annual'].fNext:=Items['p_tap_triennial'];
    //just make ssure last exists...
    Items['p_tap_triennial'].fNext:=nil;
  end;
  if TryGetValue('p_wqp_annual',T) then begin
    Items['p_wqp_annual'].fNext:=Items['p_wqp_triennial'];
    Items['p_wqp_triennial'].fNext:=Items['p_wqp_six_red'];
    //just make ssure last exists...
    Items['p_wqp_six_red'].fNext:=nil;
  end;
  if TryGetValue('p_bin1',T) then begin
    Items['p_bin1'].fNext:=Items['p_bin2'];
    Items['p_bin2'].fNext:=Items['p_bin3'];
    Items['p_bin3'].fNext:=nil;
  end;

  DirectArray:=Values.ToArray;
end;

destructor TCostVars.Destroy;
begin
  SpecialVars.Free;
  inherited;
end;

procedure TCostVars.DrawLSLReplacementRates(const aSz, aSrc, aLSL,
  aCCT, aType, aYrs : integer; const option: string; var a1: TDoubleArray);
var
  cv: TCostVar;
  i, c, ix: integer;
  curval, rawval: double;
begin
  c := Count - 1;

  if option = 'Baseline' then
  begin
    for ix := 0 to c do
    begin
      cv:=DirectArray[ix];

      if cv.fID = 'pp_lsl_replaced_one' then
      begin
         for i := 4 to aYrs do
         begin
           cv.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, true, curval, rawval,
true);
           a1[i] := rawval;
         end;
      end
    end;
  end
  else
  begin
    for ix := 0 to c do
    begin
      cv:=DirectArray[ix];
      if cv.fID = 'pp_lsl_replaced_one' then
      begin
         for i := 4 to 18 do
         begin
           cv.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, true, curval, rawval,
true);
           a1[i] := rawval;
         end;
      end
      else if cv.fID = 'pp_lsl_replaced_two' then
      begin
         for i := 19 to 24 do
         begin
           cv.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, true, curval, rawval,
true);
           a1[i] := rawval;
         end;
      end
      else if cv.fID = 'pp_lsl_replaced_three' then
      begin
         for i := 25 to aYrs do
         begin
           cv.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, true, curval, rawval,
true);
           a1[i] := rawval;
         end;
      end;
    end;
  end;
end;

procedure TCostVars.DrawPp_LSL_Replaced_Vol_Pct(const aSz, aSrc, aLSL,
  aCCT, aType, aYrs : integer; const option: string; var a1: TDoubleArray);
var
  cv: TCostVar;
  i, c, ix: integer;
  curval, rawval: double;
begin
  c := Count - 1;

  for ix := 0 to c do
  begin
    cv:=DirectArray[ix];

    if cv.fID = 'pp_lsl_replaced_vol_pct' then
    begin
      for i := 1 to aYrs do
      begin
```

```
        cv.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, true, curval, rawval, true);
        a1[i] := curval;
      end;
      break;
    end;
  end;
end;

procedure TCostVars.DrawP_Source_Chng(const aSz, aSrc, aLSL, aCCT, aType, aYrs,
nEps: integer;
  const option: string; var a1: TDoubleArray);
var
  cv: TCostVar;
  i, c, ix: integer;
  curval, rawval: double;
begin
  c := Count - 1;

  for ix := 0 to c do
  begin
    cv:=DirectArray[ix];

    if cv.fID = 'p_source_chng' then
    begin
      for i := 1 to aYrs do
      begin
        cv.GetRandomValueEp(aSz, aSrc, aLSL, aCCT, aType, nEps, true, curval,
rawval, true);
        a1[i] := curval;
      end;
      break;
    end;
  end;
end;

procedure TCostVars.DrawP_Source_Sig(const aSz, aSrc, aLSL, aCCT, aType, aYrs, nEps:
integer;
  const option: string; var a1: TDoubleArray);
var
  cv: TCostVar;
  i, c, ix: integer;
  curval, rawval: double;
begin
  c := Count - 1;

  for ix := 0 to c do
  begin
    cv:=DirectArray[ix];
    if cv.fID = 'p_source_sig' then
    begin
      for i := 1 to aYrs do
      begin
        cv.GetRandomValueEp(aSz, aSrc, aLSL, aCCT, aType, nEps, true, curval,
rawval, true);
        a1[i] := curval;
      end;
      break;
    end;
  end;
end;

procedure TCostVars.DrawP_Treat_Change(const aSz, aSrc, aLSL, aCCT, aType, aYrs,
nEps: integer;
  const option: string; var a1: TDoubleArray);
var
  cv: TCostVar;
  i, c, ix: integer;
  curval, rawval: double;
begin
  c := Count - 1;

  for ix := 0 to c do
  begin
    cv:=DirectArray[ix];

    if cv.fID = 'p_treat_change' then
    begin
      for i := 1 to aYrs do
      begin
        cv.GetRandomValueEp(aSz, aSrc, aLSL, aCCT, aType, nEps, true, curval,
rawval, true);
        a1[i] := curval;
      end;
      break;
    end;
  end;
end;

procedure TCostVars.ReadVars(Filename : string);
var
  Xls: TExcelFile;
  idname : string;
  T : TCostVar;
  r : integer;
begin
```

```
//find missing vars from database and set them to 0...
Xls := TXlsFile.Create(Filename, False);
Xls.ActiveSheetByName := 'Data Request';
{
  E 5 ID_Name
}
for r := 3 to Xls.RowCount do begin
  IDName:=Xls.GetStringFromCell(r, 5);
  if (IDName <> '') then begin
    if TryGetValue(IDName,T) then continue;
    T:=TCostVar.CreateZero(IDName);
    Add(T.fID,T);
    inc(NumVars);
  end;
end;
FreeAndNil(Xls);

//Add special variables
T:=TCostVar.CreateZero('numb_ep');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('pws_cct');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('p_fail');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('num_lsl_replace');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('meet_lslr_goal');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('pws_first_ale');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('pws_sw');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('pws_gw');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('pws_pop');
Add(T.fID,T);
inc(NumVars);
```

```
T:=TCostVar.CreateZero('hh_remain_lsl');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('b_wqp_chng_adj');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_copper_agg_adj');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_source_chng');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_cct_guid_chng');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_treat_chng');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_lead_agg_inst');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_copper_agg_inst');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_source_chng');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_treat_chng');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('b_adjust_cct_copper');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_lead_ale');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_copper_ale');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_treat');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('cct_existing_cost');
Add(T.fID,T);
inc(NumVars);
```

```
T:=TCostVar.CreateZero('cct_modify_cost');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_install_cost');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_findfix_cost');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_modify_cost_umra');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_install_cost_umra');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_findfix_cost_umra');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_modify_cost_umra_om');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_install_cost_umra_om');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_findfix_cost_umra_om');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('cct_modify_cost_p');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_install_cost_p');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('cct_findfix_cost_p');
Add(T.fID,T);
inc(NumVars);


T:=TCostVar.CreateZero('b_adjust_cct_sal_p1');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_sal_p2');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_sal_p3');
Add(T.fID,T);
inc(NumVars);
```

```
T:=TCostVar.CreateZero('b_install_cct_sal_p1');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_sal_p2');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_sal_p3');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('b_adjust_cct_lead_plat_1');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_lead_plat_2');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_lead_plat_3');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_copper_agg_adj_plat');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_source_chng_plat');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_treat_chng_plat');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_lead_plat_1');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_lead_plat_2');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_lead_plat_3');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_copper_agg_inst_plat');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('b_modify_cct_50_lsl');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_50_lsl');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_lead_green_1');
```

```
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_lead_green_2');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_lead_green_3');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_lead_green_1');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_lead_green_2');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_lead_green_3');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('adjust_cct');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('install_cct');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_copper_agg_adj_green');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_copper_agg_inst_green');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_source_chng_green');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_source_chng_green');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_install_cct_treat_chng_green');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_adjust_cct_treat_chng_green');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('num_hh_per_connect');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('pws_fail');
Add(T.fID,T);
inc(NumVars);
```

```
{$IFDEF NDWAC_TEST}
  T:=TCostVar.CreateZero('b_cct_guid_chng_five');
  Add(T.fID,T);
  inc(NumVars);
{$ENDIF}

  T:=TCostVar.CreateZero('b_install_cct_lead_ale_one');
  Add(T.fID,T);
  inc(NumVars);
  T:=TCostVar.CreateZero('b_install_cct_lead_ale_two');
  Add(T.fID,T);
  inc(NumVars);
  T:=TCostVar.CreateZero('b_install_cct_lead_ale_three');
  Add(T.fID,T);
  inc(NumVars);

  T:=TCostVar.CreateZero('b_cct_sanitary_survey');
  Add(T.fID,T);
  inc(NumVars);

  T:=TCostVar.CreateZero('num_lsl_paper');
  Add(T.fID,T);
  inc(NumVars);

  T:=TCostVar.CreateZero('b_modify_cct');
  Add(T.fID,T);
  inc(NumVars);
  T:=TCostVar.CreateZero('b_install_cct');
  Add(T.fID,T);
  inc(NumVars);

  T:=TCostVar.CreateZero('b_modify_cct_mc');
  Add(T.fID,T);
  inc(NumVars);
  T:=TCostVar.CreateZero('b_install_cct_mc');
  Add(T.fID,T);
  inc(NumVars);

  T:=TCostVar.CreateZero('b_install_pou');
  Add(T.fID,T);
  inc(NumVars);
  T:=TCostVar.CreateZero('system_pou');
  Add(T.fID,T);
  inc(NumVars);

  T:=TCostVar.CreateZero('b_findfix');
  Add(T.fID,T);
  inc(NumVars);
```

```
T:=TCostVar.CreateZero('b_lslr_study');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_pou_study');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_lslr_mand');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_lslr_vol');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_lslr_requested');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('school_1a');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('school_1b');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('school_3a');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('school_3b');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('school_3c');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('school_3d');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('school_5a');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('school_5b');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('b_cct_study_install');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_cct_study_rec_install');
Add(T.fID,T);
inc(NumVars);
```

```
T:=TCostVar.CreateZero('b_state_cct_treatment_install');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_cct_study_mod');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_cct_study_rec_mod');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_state_cct_treatment_mod');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('fail_nm1');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('fail_nm2');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('b_cct_study_rec_mod_tl');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_cct_study_mod_tl');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_modify_cct_tl');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_state_cct_treatment_mod_tl');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_cct_study_rec_mod_al');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_cct_study_mod_al');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_modify_cct_al');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_state_cct_treatment_mod_al');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('numb_wqp_sites_added');
Add(T.fID,T);
inc(NumVars);
```

```
T:=TCostVar.CreateZero('numb_wqp_sites_added_prev');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('num_lsl_requested');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('numb_second_schools_pub');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('numb_elem_schools_pub');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('numb_second_schools_priv');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('numb_elem_schools_priv');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('p_grandfather_opt_pub');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('p_grandfather_opt_priv');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('p_grandfather_mand_pub');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('p_grandfather_mand_priv');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('b_state_one');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('b_state_two');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('num_lsl_remain');
Add(T.fID,T);
inc(NumVars);
T:=TCostVar.CreateZero('num_paper_remain');
Add(T.fID,T);
inc(NumVars);

T:=TCostVar.CreateZero('p_grandfather_opt_child');
```

```
  Add(T.fID,T);
  inc(NumVars);
  T:=TCostVar.CreateZero('p_grandfather_mand_child');
  Add(T.fID,T);
  inc(NumVars);
end;


procedure TCostVars.ResetRandomSeeds(PWSSeed: integer);
var v : TCostVar;
    ix,c : integer;
begin
  c:=Count-1;
  //for v in Values do begin
  for ix:=0 to c do begin
    v:=DirectArray[ix];
    v.SetCustomSeed(PWSSeed);
  end;
end;

procedure TCostVars.FillValueArray(var a1, a2: TDoubleArray; const aSz, aSrc, aLSL,
aCCT, aType, Yr, Bp1, Bp2: integer;
  const SetProbsTo01, ForceDraw: boolean; O : TDictionary<string,double>;
IsBaseline: boolean);
var v,v2 : TCostVar;
    i,c,ix : integer;
    d,r : double;
    curval, rawval: double;
    fnd: boolean;
begin
  i:=0;
  if length(a1)<NumVars then setlength(a1,NumVars);
  if length(a2)<NumVars then setlength(a2,NumVars);
  c:=Count-1;

  //for v in Values do begin
  for ix:=0 to c do begin
    v:=DirectArray[ix];
    if (Yr=1) or (v.fChangesYearly) or (ForceDraw) then begin
      if v.fID = 'dist_lead_base_bin1' then begin
        fnd := false;
        while not fnd do begin
          v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval,
rawval, IsBaseline);
          curval := curval * 1000;
          v.fCurValue := curval;
          v.fRawValue := rawval * 1000;
          if curval > Bp2 then fnd := true;
        end;
      end else
      if v.fID = 'dist_lead_base_bin2' then begin
        fnd := false;
        while not fnd do begin
          v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, 
rawval, IsBaseline);
          curval := curval * 1000;
          v.fCurValue := curval;
          v.fRawValue := rawval * 1000;
          if (curval > Bp1) and (curval <= Bp2) then fnd := true;
        end;
      end else
      if v.fID = 'dist_lead_base_bin3' then begin
        fnd := false;
        while not fnd do begin
          v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, 
rawval, IsBaseline);
          curval := curval * 1000;
          v.fCurValue := curval;
          v.fRawValue := rawval * 1000;
          if curval <= Bp1 then fnd := true;
        end;
      end else
        v.GetRandomValue(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, rawval, 
IsBaseline);

      a1[i] := curval;
      a2[i] := rawval;
    end;
    inc(i);
  end;

  //now we have all values, lets reset the dependent ones...
  //for v in Values do begin
  for ix:=0 to c do begin
    v:=DirectArray[ix];
    if (v.fCurValue=1) and (Assigned(v.fNext)) then begin
      v2:=v.fNext;
      repeat
        v2.fCurValue:=0;
        a1[v2.fIDX]:=0;
        a2[v2.fIDX]:=0;
        v2:=v2.fNext;
      until not Assigned(v2);
    end;
  end;
  if not assigned(O) then exit;
  if (not ForceDraw) and (Yr>1) then exit;

  i:=0;
  //for v in Values do begin
  for ix:=0 to c do begin
    v:=DirectArray[ix];
    if Yr > 1 then begin
     if v.fByLSL or v.fByCCT or v.fChangesYearly then begin
       inc(i);
       continue;
     end;
    end;

    if v.fImSpecial then begin
      if IsBaseline then r:=RG.GetB else r:=RG.GetS;
      if not O.TryGetValue(v.fIDR,d) then begin
        //this shouldnt happen right?
      end else begin
        a2[i] := d;
        if R<=d then a1[i]:=1 else a1[i]:=0;
      end;
    end else begin
      if O.TryGetValue(v.fID,d) then
        a1[i] := d;
      if O.TryGetValue(v.fIDR,d) then
        a2[i] := d;
    end;
    inc(i);
  end;
end;

procedure TCostVars.FillValueArray2(var a1, a2: TDoubleArray; const aSz, aSrc,
  aLSL, aCCT, aType, Yr, Bp1, Bp2: integer; const SetProbsTo01, ForceDraw: boolean;
  const BaselineCols, BaselineData: TStringList);
var v,v2 : TCostVar;
    i : integer;
    curval, rawval: double;
    fnd: boolean;
begin
  i:=0;
  if length(a1)<NumVars then setlength(a1,NumVars);
  if length(a2)<NumVars then setlength(a2,NumVars);
  for v in Values do begin
    if (Yr=1) or (v.fChangesYearly) or (ForceDraw) then
    begin
      if v.fID = 'dist_lead_base_bin1' then
      begin
```

```
        fnd := false;
        while not fnd do
        begin
          v.GetRandomValue2(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, 
rawval,
                            BaselineCols, BaselineData);
          if curval > Bp2 then fnd := true;
        end;
      end
      else
      if v.fID = 'dist_lead_base_bin2' then
      begin
        fnd := false;
        while not fnd do
        begin
          v.GetRandomValue2(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, 
rawval,
                            BaselineCols, BaselineData);
          if (curval > Bp1) and (curval <= Bp2) then fnd := true;
        end;
      end
      else
      if v.fID = 'dist_lead_base_bin3' then
      begin
        fnd := false;
        while not fnd do
        begin
          v.GetRandomValue2(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, 
rawval,
                            BaselineCols, BaselineData);
          if curval <= Bp1 then fnd := true;
        end;
      end
      else
        v.GetRandomValue2(aSz, aSrc, aLSL, aCCT, aType, SetProbsTo01, curval, 
rawval,
                          BaselineCols, BaselineData);

      a1[i] := curval;
      a2[i] := rawval;
    end;
    inc(i);
  end;
  //now we have all values, lets reset the dependent ones...
  for v in Values do begin
    if (v.fCurValue=1) and (Assigned(v.fNext)) then begin
      v2:=v.fNext;
      repeat
```

```
        v2.fCurValue:=0;
        a1[v2.fIDX]:=0;
        a2[v2.fIDX]:=0;
        v2:=v2.fNext;
      until not Assigned(v2);
    end;
  end;
end;



initialization
  NoRandom:=False;
  UserSeeds:=True;
  RG:=TRandomGiver.Create;
  RG.Open;
finalization
  RG.Free;
end.
```