```
unit LCRCosts;

interface

uses dialogs, Classes, SysUtils, StrUtils, LCRGlobals, LCRConfig,
     LCRMetricCollector, SafewaterUncertBucket, CostingSteps,
     LCRAggregationConsts,LCRCostVars,CodeSiteLogging,
     DB, ADODB, Generics.Collections, CCTCostEquations, StateSchoolSampData;
type

  TAggInputRec = record
    AggName: string;
    BaselineValue: double;
    OptionValue: double;
    Difference: double;
    Bidx,BidxP,
    SIdx,SIdxp : integer;
    AggType: string;
    IsMain : boolean;
  end;

  TLCRCosts = class(TObject)
    private

      BaseCostVars, ScenCostVars : TCostVars;
      //these will hold our final aggregated values (Scen-Baseline)
      fResults, fResultsPWS : array[0..10000] of double;
      fResultsCapital: array[0..2] of double;
      fResultsICR: array[0..10000] of double;

      DummyProb,Flip : double;

      CCTCostEquations: TCCTCostEquations;

      AggInput: TAggInputRec;

      AggregatedResults: TDictionary<string, TAggInputRec>;

      HasCCTCnt, NoCCTCnt, HasLSLCnt, NoLSLCnt: integer;
      counter: integer;

      SystemAFlowBaseline, SystemAFlowOption: double;
      POTWCost: double;

      function CostByStatePWSCount(cost: double): double;
      procedure UMRACosts;
      procedure CalculateCustomMetrics(hh_consumption: double; VLSystem: boolean);
      procedure VLSEpLoop(CostingData: TCostGenRec; option: string; SchoolSampData:
```

```
TSchoolSampDataRec);

      function GetVOld(const s : string; const base : boolean) : double;
      function GetV(const s : string) : double;
      function GetVESingle(const s : TArray<string>; Baseline : boolean) : double;
      function GetVE(const s : TArray<string>) : double;
      procedure ProxyCost;
    public
      Config: TLCRConfig;
      Outputs: TMetricList;

      PWSCount: double;
      LSLReplacement, LSLReplacementRequested: double;
      LSLReplacementPopulation, LSLReplacementPopRequested: double;
      LSLReplacementMandatory, LSLReplacementVoluntary: double;
      LSLReplacementPopMandatory, LSLReplacementPopVoluntary: double;
      AddModifyCCT: double;
      AddModifyCCTPopulation: double;
      LSLReplaced, LSLReplacedRequested: double;
      LSLReplacedMandatory: double;
      LSLReplacedVoluntary: double;

      CCTInstalled: double;
      CCTInstalledPopulation: double;
      CCTAdjusted, CCTAdjusted_ale, CCTAdjusted_tle: double;
      CCTAdjustedPopulation, CCTAdjustedPopulation_ale, CCTAdjustedPopulation_tle:
double;
      CCTExisting: double;
      CCTExistingPopulation: double;
      FindAndFixCost: double;
      FindAndFixCostPopulation: double;
      POUInstalled: double;
      POUInstalledPopulation: double;

      TotalPWSCost, TotalHHCost, TotalStateCost, TotalRuleCost, TotalCostCap,
TotalCostCapPrx,
      TotalHHCostPWS : double;
      TotalICRCost_1, TotalICRCost_2, TotalICRCost_3, TotalICRCost_4,
      TotalICRCost_10 : double;
      TotalICRHours_1, TotalICRHours_2, TotalICRHours_3, TotalICRHours_4,
      TotalICRHours_10 : double;
      TotalICRHoursState_1, TotalICRHoursState_2, TotalICRHoursState_3,
      TotalICRHoursState_4, TotalICRHoursState_10 : double;
      TotalICRCostState_1, TotalICRCostState_2, TotalICRCostState_3,
      TotalICRCostState_4, TotalICRCostState_10 : double;

      TotalLSLR,TotalLSLRPWS : double;
      TotalPWSSamp,TotalPWSSampPWS,TotalSampleAdmin,
```

```
      TotalPWSLSLR,TotalPWSLSLRPWS,

TotalPWSLSLRMandatory,TotalPWSLSLRVoluntary,TotalPWSLSLRPWSMandatory,TotalPWSLSLRPWS
Voluntary,
      TotalLSLRPOU,TotalPWSLSLRRequested,TotalPWSLSLRPWSRequested,
      TotalCCT,TotalCCTPWS,
      TotalEDU,TotalEDUPWS : double;
      CostRevenueRatio,CRGT1,CRGT3 : double;
      LSLReplacementHouseholdTotal: double;

      TotalPWSCostUPrivate,TotalPWSCostUPublic : array[1..100] of double;
      TotalStateCostU,TotalLSLHHCostU : array[1..100] of double;

      //adding these as output checks for umra
      aTotalPWSCostUPrivate,aTotalPWSCostUPublic :  double;
      aTotalStateCostU,aTotalLSLHHCostU : double;


      prerule_ploading_lbs_5, prerule_ploading_lbs_15, prerule_ploading_lbs_25,
prerule_ploading_lbs_35: double;
      postrule_ploading_lbs_5, postrule_ploading_lbs_15, postrule_ploading_lbs_25,
postrule_ploading_lbs_35: double;
      incr_ploading_lbs_5, incr_ploading_lbs_15, incr_ploading_lbs_25,
incr_ploading_lbs_35: double;
      count_incr_ploading_lbs_5, count_incr_ploading_lbs_15,
count_incr_ploading_lbs_25, count_incr_ploading_lbs_35: double;

      CostingData : TCostGenRec;
      BAddCostingData,SAddCostingData : TAddCostGenRec;
      VLSEpWorkbook: TVLSEpWorkbookRec;

      nDR: integer;

      strLeadConcentrationsS,strLeadConcentrationsB : TBufferedFileStream;
      //making these public to get to the bins...
      BaseCostSteps, ScenCostSteps: TCostingSteps;

      constructor Create(aConfig: TLCRConfig; aOutput: TMetricList; aUncertainty:
TUncertaintyStudy);
      destructor Destroy; override;

      procedure SetCCTCostEquationsData;

      procedure GenerateCosts(RndSeed : integer; slProxies: TStringList;
SchoolSampData: TSchoolSampDataRec);
      procedure ResetCosts;
      procedure StateCosts;
      function GetTotalEvaluations : int64;
```

```
      function GetTotalCompiledEvaluations : int64;
  end;

implementation

uses Math, VCL.FlexCel.Core, FlexCel.XlsAdapter, System.Variants;

{ TLCRCosts }

function TLCRCosts.GetVOld(const s : string; const base : boolean) : double;
begin
  Result:=0;
  if  AggregatedResults.TryGetValue(s,AggInput) then
   if base then
     Result:=AggInput.BaselineValue
   else
     Result:=AggInput.OptionValue;
end;

function TLCRCosts.GetV(const s : string) : double;
begin
  Result:=0;
  if  AggregatedResults.TryGetValue(s,AggInput) then
    Result := (AggInput.OptionValue - AggInput.BaselineValue) * Flip;
end;

function TLCRCosts.GetVESingle(const s : TArray<string>; Baseline : boolean) :
double;
var i : integer;
begin
  Result:=0;
  for i:=0 to high(s) do begin
    if AggregatedResults.TryGetValue(s[i],AggInput) then begin
      if Baseline then
        Result:=Result + AggInput.BaselineValue
      else
        Result:=Result + AggInput.OptionValue;
    end;
  end;
end;

function TLCRCosts.GetVE(const s : TArray<string>) : double;
var i : integer;
begin
  Result:=0;
  for i:=0 to high(s) do begin
    if AggregatedResults.TryGetValue(s[i],AggInput) then
      Result:=Result + (AggInput.OptionValue - AggInput.BaselineValue) * Flip;
```

```
  end;
end;


procedure TLCRCosts.ProxyCost;
var i,sx : integer;
begin
  TotalCostCapPrx := 0 ;
  for sx:=0 to high(acTotalCostCap) do begin
    if not AggregatedResults.TryGetValue(acTotalCostCap[sx],AggInput) then continue;
    for i:=1 to Config.YearsOfAnalysis do begin
      if AggInput.Sidx>-1 then begin
        TotalCostCapPrx := TotalCostCapPrx + ScenCostSteps.Values2Y[i,AggInput.SIdx]
                                           / PreCalcDiscRate[i-1];
      end;
      if AggInput.Bidx>-1 then begin
        TotalCostCapPrx := TotalCostCapPrx - BaseCostSteps.Values2Y[i,AggInput.BIdx]
                                           / PreCalcDiscRate[i-1];
      end;
    end;
  end;
end;


procedure TLCRCosts.UMRACosts;
var i,sx,bstx,sstx,blx,slx : integer;
    av,ad : double;
begin
  //this is done adhoc until we introcude yearly metrics generally
  fillchar(TotalPWSCostUPrivate,sizeof(TotalPWSCostUPrivate),0);
  fillchar(TotalPWSCostUPublic,sizeof(TotalPWSCostUPublic),0);
  fillchar(TotalStateCostU,sizeof(TotalStateCostU),0);
  fillchar(TotalLSLHHCostU,sizeof(TotalLSLHHCostU),0);
  aTotalPWSCostUPrivate := 0; aTotalPWSCostUPublic := 0;
  aTotalStateCostU := 0; aTotalLSLHHCostU := 0;

  for sx:=0 to high(acTotalPWSCostUMRA) do begin
    if  not AggregatedResults.TryGetValue(acTotalPWSCostUMRA[sx],AggInput) then
continue;
    for i:=1 to Config.YearsOfOutput do begin
      if AggInput.Sidx>-1 then begin
        if (CostingData.Ownership=1) then
          TotalPWSCostUPrivate[i]:=TotalPWSCostUPrivate[i] +
ScenCostSteps.Values2Y[i,AggInput.SIdx]
        else
          TotalPWSCostUPublic[i]:=TotalPWSCostUPublic[i] +
ScenCostSteps.Values2Y[i,AggInput.SIdx];
      end;
```

```
      if AggInput.Bidx>-1 then begin
        if (CostingData.Ownership=1) then
          TotalPWSCostUPrivate[i]:=TotalPWSCostUPrivate[i] -
BaseCostSteps.Values2Y[i,AggInput.BIdx]
        else
          TotalPWSCostUPublic[i]:=TotalPWSCostUPublic[i] -
BaseCostSteps.Values2Y[i,AggInput.BIdx];
      end;
    end;
  end;

  blx:=-1;
  slx:=-1;
  if AggregatedResults.TryGetValue('LSL Replacement Household Voluntary
Total',AggInput) or
     AggregatedResults.TryGetValue('LSL Replacement Household Mandatory
Total',AggInput) then begin
    blx:=AggInput.BIdx;
    slx:=AggInput.SIdx;
  end;
  bstx:=-1;
  sstx:=-1;
  if AggregatedResults.TryGetValue('Total State Admin',AggInput) then begin
    bstx:=AggInput.BIdx;
    sstx:=AggInput.SIdx;
  end;

  for i:=1 to Config.YearsOfOutput do begin
     if slx>-1 then
       TotalLSLHHCostU[i]:=TotalLSLHHCostU[i] + ScenCostSteps.Values2Y[i,slx];
     if blx>-1 then
       TotalLSLHHCostU[i]:=TotalLSLHHCostU[i] - BaseCostSteps.Values2Y[i,blx];

     if sstx>-1 then
       TotalStateCostU[i]:=TotalStateCostU[i] + ScenCostSteps.Values2Y[i,sstx];
     if bstx>-1 then
       TotalStateCostU[i]:=TotalStateCostU[i] - BaseCostSteps.Values2Y[i,bstx];
  end;

  //now correct for baseline only run...
  for i:=1 to Config.YearsOfOutput do begin
    TotalPWSCostUPrivate[i]:=TotalPWSCostUPrivate[i] * Flip;
    TotalPWSCostUPublic[i]:=TotalPWSCostUPublic[i] * Flip;
    TotalStateCostU[i]:=TotalStateCostU[i] * Flip;
    TotalLSLHHCostU[i]:=TotalLSLHHCostU[i] * Flip;
  end;

  //calculate umracheck variables
```

```
  av:=(Config.DiscountRate / (1 - IntPower((1 +
Config.DiscountRate),-Config.YearsOfOutput)));
  for i:=1 to Config.YearsOfOutput do begin
    ad :=  1/intpower((1+Config.DiscountRate),i - 1);
    aTotalPWSCostUPrivate:=aTotalPWSCostUPrivate + ((ad * TotalPWSCostUPrivate[i]) *
av ) ;
    aTotalPWSCostUPublic:=aTotalPWSCostUPublic + ((ad * TotalPWSCostUPublic[i]) * av
) ;
    aTotalStateCostU:=aTotalStateCostU + ((ad * TotalStateCostU[i]) * av ) ;
    aTotalLSLHHCostU:=aTotalLSLHHCostU + ((ad * TotalLSLHHCostU[i]) * av ) ;
  end;

end;

procedure TLCRCosts.CalculateCustomMetrics(hh_consumption: double; VLSystem:
boolean);
var v,v2,v3,VLFLow : double;
    LSRT : double;
begin

  LSRT:=GetV('LSL Replacement Household Voluntary Total') + GetV('LSL Replacement
Household Mandatory Total');
  TotalPWSCost := GetVE(acTotalPWSCost);
  TotalPWSSamp := GetVE(acTotalPWSSamp);
  TotalPWSSampPWS  :=  GetVE(acTotalPWSSampPWS);
  TotalPWSLSLR := GetVE(acTotalPWSLSLR);
  TotalPWSLSLRMandatory := GetVE(acPWSLSLRMandatory);
  TotalPWSLSLRVoluntary := GetVE(acPWSLSLRVoluntary);
  TotalPWSLSLRRequested := GetVE(acPWSLSLRRequested);
  TotalLSLR := TotalPWSLSLR + GetV('LSL Replacement Household Voluntary Total') +
GetV('LSL Replacement Household Mandatory Total');
  TotalPWSLSLRPWS := GetVE(acTotalPWSLSLRPWS);
  TotalPWSLSLRPWSMandatory := GetVE(acPWSLSLRMandatoryPWS);
  TotalPWSLSLRPWSVoluntary := GetVE(acPWSLSLRVoluntaryPWS);
  TotalPWSLSLRPWSRequested := GetVE(acPWSLSLRRequestedPWS);
  TotalLSLRPWS := TotalPWSLSLRPWS + GetV('LSL Replacement Household Voluntary
Total') + GetV('LSL Replacement Household Mandatory Total');
  TotalCCT := GetVE(acTotalCCT);
  TotalCCTPWS := GetVE(acTotalCCTPWS);
  TotalEDU := GetV('System Lead Public Education Total');
  TotalEDUPWS := GetV('System Lead Public Education Total_PWS');

  TotalCostCap := GetVE(acTotalCostCap);

  TotalStateCost := GetV('Total State Admin');
  TotalRuleCost := TotalPWSCost + TotalStateCost + GetV('LSL Replacement Household
Voluntary Total') + GetV('LSL Replacement Household Mandatory Total') +
                   POTWCost;
```

```
  LSLReplacementHouseholdTotal := LSRT;

  TotalICRCost_1 := GetVE(acTotalICRCost_1);
  TotalICRCost_2 := GetVE(acTotalICRCost_2);
  TotalICRCost_3 := GetVE(acTotalICRCost_3);
  TotalICRCost_4 := GetVE(acTotalICRCost_4);
  TotalICRCost_10 := GetVE(acTotalICRCost_10);

  TotalICRCostState_1 := GetVE(acTotalICRCostState_1);
  TotalICRCostState_2 := GetVE(acTotalICRCostState_2);
  TotalICRCostState_3 := GetVE(acTotalICRCostState_3);
  TotalICRCostState_4 := GetVE(acTotalICRCostState_4);
  TotalICRCostState_10 := GetVE(acTotalICRCostState_10);

  TotalICRHours_1 := GetVE(acTotalICRHours_1);
  TotalICRHours_2 := GetVE(acTotalICRHours_2);
  TotalICRHours_3 := GetVE(acTotalICRHours_3);
  TotalICRHours_4 := GetVE(acTotalICRHours_4);
  TotalICRHours_10 := GetVE(acTotalICRHours_10);


  TotalICRHoursState_1 := GetVE(acTotalICRHoursState_1);
  TotalICRHoursState_2 := GetVE(acTotalICRHoursState_2);
  TotalICRHoursState_3 := GetVE(acTotalICRHoursState_3);
  TotalICRHoursState_4 := GetVE(acTotalICRHoursState_4);
  TotalICRHoursState_10 := GetVE(acTotalICRHoursState_10);

  TotalSampleAdmin:=TotalPWSSamp+TotalStateCost;
  TotalLSLRPOU:=TotalPWSLSLR+LSRT;

  if VLSystem then  begin
    //todo flows do not appear different...   This is a hack to avoid cheking run
type.  Fix...
    VLFLow:=max(SystemAFlowBaseline,SystemAFlowOption);
    v:=TotalCostCap / (VLFlow * 1000000);
    v2:=v * hh_consumption;
    v:=TotalPWSCost / (VLFlow * 1000000 );
    v3:=v * hh_consumption;
  end else begin
    //cost per gallon....
    v:=TotalCostCap / ( (CostingData.AFlowEP * CostingData.EntryPoints) * 1000000 );
    //cost per hh....
    v2:=v * hh_consumption;
    //cost per gallon....
    v:=TotalPWSCost / ( (CostingData.AFlowEP * CostingData.EntryPoints) * 1000000 );
    //cost per hh....
    v3:=v * hh_consumption;
```

```
  end;

  TotalHHCost := v3 + (GetV('LSL Replacement Household Voluntary Total')+GetV('LSL
Replacement Household Mandatory Total')) / (CostingData.Population / 2.59);
  TotalHHCostPWS := v2 + (GetV('LSL Replacement Household Voluntary
Total_PWS')+GetV('LSL Replacement Household Mandatory Total_PWS')) /
(CostingData.Population / 2.59);

  CostRevenueRatio:=TotalCostCap / CostingData.PWSAnnualRevenue;
  if Config.RunDifference then begin
    CRGT1:=0;
    CRGT3:=0;
    v := GetVESingle(acTotalCostCap,False);
    v2 := GetVESingle(acTotalCostCap,True);
    v2 := (v - v2) / CostingData.PWSAnnualRevenue;
    if v2>=0.01 then CRGT1:=1;
    if v2>=0.03 then CRGT3:=1;
  end else begin
    if CostRevenueRatio>=0.01 then CRGT1:=1 else CRGT1:=0;
    if CostRevenueRatio>=0.03 then CRGT3:=1 else CRGT3:=0;
  end;

  UMRACosts;
end;

function TLCRCosts.CostByStatePWSCount(cost: double): double;
var
  i: integer;
  totalcost: double;
begin
  totalcost := 0;
  for i := 0 to High(Config.StateDataArray) do
  begin
    totalcost := totalcost + (cost * Config.StateDataArray[i].PWSCount);
  end;
  Result := totalcost;
end;

constructor TLCRCosts.Create(aConfig: TLCRConfig; aOutput: TMetricList;
aUncertainty: TUncertaintyStudy);
var
  i: integer;
  ist : string;
  ADOCon : TADOCOnnection;
  qDesc, qData : TADOQuery;
  DictItem: TPair<string, TAggInputRec>;
begin
  Config := aConfig;
```

```
  Outputs := aOutput;
  DummyProb := 1;
  nDR := High(Config.DiscRates);

  ADOCon:=TADOCOnnection.Create(nil);
  ADOCon.ConnectionString:=format(ADOConStr,[aConfig.BaseVarData]);
  qDesc:=TADOQuery.Create(nil);
  qDesc.Connection:=ADOCon;
  qData:=TADOQuery.Create(nil);
  qData.Connection:=ADOCon;
  qDesc.SQL.Add('select * from InputDesc');
  qData.SQL.Add('select * from InputValues');

  qDesc.Open;
  qData.Open;

BaseCostVars:=TCostVars.Create(qDesc,qData,ChangeFileExt(aConfig.BaseVarData,'.xlsm'
));
  qData.Close;
  qDesc.Close;
  AdoCon.Close;

  ADOCon.ConnectionString:=format(ADOConStr,[aConfig.ScenVarData]);
  qData.Open;
  qDesc.Open;

ScenCostVars:=TCostVars.Create(qDesc,qData,ChangeFileExt(aConfig.ScenVarData,'.xlsm'
), BaseCostVars);
  AdoCon.Close;
  qData.Free;
  qDesc.Free;
  ADOCon.Free;

  AggregatedResults := TDictionary<string, TAggInputRec>.Create;
  AggInput.AggName := '';
  AggInput.BaselineValue := 0;
  AggInput.OptionValue := 0;
  AggInput.Difference := 0;
  AggInput.SIdx:=-1;
  AggInput.SIdxp:=-1;
  AggInput.BIdx:=-1;
  AggInput.BIdxp:=-1;


  BaseCostSteps := TCostingSteps.Create(BaseCostVars,aConfig.BaseCostSteps, '',
Config.YearsOfAnalysis,
                                        aConfig.YearsOfOutput,True);
  BaseCostSteps.DiscRate:=aConfig.DiscountRate;
```

```
  BaseCostSteps.Config := aConfig;
  aConfig.Log.Text:='Baseline Costing Step Errors:';
  aConfig.Log.Strings:=BaseCostSteps.Errors;

  for i := 0 to Length(BaseCostSteps.AggInputs2) - 1 do begin
    AggInput.AggName := BaseCostSteps.AggInputs2[i];
    AggInput.IsMain := not ((ContainsText(AggInput.AggName,'_hours')) or
                            (ContainsText(AggInput.AggName,'_labor')) or
                            (ContainsText(AggInput.AggName,'_om')));

    AggInput.AggType := '2';
    AggInput.Bidx:=i;
    AggregatedResults.AddOrSetValue(BaseCostSteps.AggInputs2[i],AggInput);

    AggInput.AggName := Trim(BaseCostSteps.AggInputs2[i]) + '_PWS';
    AggInput.AggType := '2PWS';
    AggInput.Bidxp:=i;
    AggregatedResults.AddOrSetValue(AggInput.AggName,AggInput);
  end;

  for i := 0 to Length(BaseCostSteps.AggICR) - 1 do begin
    AggInput.AggName := BaseCostSteps.AggICR[i];
    AggInput.IsMain := not ((ContainsText(AggInput.AggName,'_hours')) or
                            (ContainsText(AggInput.AggName,'_labor')) or
                            (ContainsText(AggInput.AggName,'_om')));

    AggInput.AggType := 'ICR';
    AggInput.Bidx:=i;
    AggregatedResults.AddOrSetValue(BaseCostSteps.AggICR[i],AggInput);
  end;

  ScenCostSteps := TCostingSteps.Create(ScenCostVars,aConfig.ScenCostSteps, '',
Config.YearsOfAnalysis,
                                        aConfig.YearsOfOutput, False);
  ScenCostSteps.DiscRate:=aConfig.DiscountRate;
  ScenCostSteps.Config := aConfig;
  aConfig.Log.Text:='Scenario Costing Step Errors:';
  aConfig.Log.Strings:=ScenCostSteps.Errors;

  for i := 0 to Length(ScenCostSteps.AggInputs2) - 1 do begin
    if not(AggregatedResults.TryGetValue(ScenCostSteps.AggInputs2[i],AggInput)) then
      fillchar(AggInput,sizeof(AggInput),0);

    AggInput.AggName := ScenCostSteps.AggInputs2[i];
    AggInput.IsMain := not ((ContainsText(AggInput.AggName,'_hours')) or
                            (ContainsText(AggInput.AggName,'_labor')) or
                            (ContainsText(AggInput.AggName,'_om')));
```

```
    AggInput.AggType := '2';
    AggInput.Sidx:=i;
    AggregatedResults.AddOrSetValue(ScenCostSteps.AggInputs2[i],AggInput);

    if not(AggregatedResults.TryGetValue(Trim(ScenCostSteps.AggInputs2[i]) +
'_PWS',AggInput)) then
      fillchar(AggInput,sizeof(AggInput),0);

    AggInput.AggName := Trim(ScenCostSteps.AggInputs2[i]) + '_PWS';
    AggInput.IsMain := not ((ContainsText(AggInput.AggName,'_hours')) or
                          (ContainsText(AggInput.AggName,'_labor')) or
                          (ContainsText(AggInput.AggName,'_om')));
    AggInput.AggType := '2PWS';
    AggInput.Sidxp:=i;
    AggregatedResults.AddOrSetValue(AggInput.AggName,AggInput);
  end;

  for i := 0 to Length(ScenCostSteps.AggICR) - 1 do begin
    if not(AggregatedResults.TryGetValue(ScenCostSteps.AggICR[i],AggInput)) then
      fillchar(AggInput,sizeof(AggInput),0);

    AggInput.AggName := ScenCostSteps.AggICR[i];
    AggInput.IsMain := not ((ContainsText(AggInput.AggName,'_hours')) or
                          (ContainsText(AggInput.AggName,'_labor')) or
                          (ContainsText(AggInput.AggName,'_om')));
    AggInput.AggType := 'ICR';
    AggInput.Sidx:=i;
    AggregatedResults.AddOrSetValue(ScenCostSteps.AggICR[i],AggInput);
  end;

  Outputs.AddOutputMetric(@PWSCount,@DummyProb,nil,'PWS
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacement,@DummyProb,nil,'LSL Replacement
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementMandatory,@DummyProb,nil,'LSL Replacement
Count Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementVoluntary,@DummyProb,nil,'LSL Replacement
Count Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementPopulation,@DummyProb,nil,'LSL Replacement
Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementPopMandatory,@DummyProb,nil,'LSL
Replacement Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementPopVoluntary,@DummyProb,nil,'LSL
Replacement Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementRequested,@DummyProb,nil,'LSL Replacement
Count Requested',mtInfo,False,False,False,Config.OptionName,0,True);
```

```
  Outputs.AddOutputMetric(@LSLReplacementPopRequested,@DummyProb,nil,'LSL
Replacement Population
Requested',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplaced,@DummyProb,nil,'LSL
Replaced',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacedMandatory,@DummyProb,nil,'LSL Replaced
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacedVoluntary,@DummyProb,nil,'LSL Replaced
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacedRequested,@DummyProb,nil,'LSL Replaced
Requested',mtInfo,False,False,False,Config.OptionName,0,True);

  Outputs.AddOutputMetric(@CCTInstalled,@DummyProb,nil,'CCT Installed
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTInstalledPopulation,@DummyProb,nil,'CCT Installed
Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTAdjusted,@DummyProb,nil,'CCT Modified
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTAdjusted_ale,@DummyProb,nil,'CCT Modified ALE
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTAdjusted_tle,@DummyProb,nil,'CCT Modified TLE
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTAdjustedPopulation,@DummyProb,nil,'CCT Modified
Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTAdjustedPopulation_ale,@DummyProb,nil,'CCT Modified
ALE Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTAdjustedPopulation_tle,@DummyProb,nil,'CCT Modified
TLE Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTExisting,@DummyProb,nil,'CCT Existing
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@CCTExistingPopulation,@DummyProb,nil,'CCT Existing
Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@FindAndFixCost,@DummyProb,nil,'Find and Fix Cost
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@FindAndFixCostPopulation,@DummyProb,nil,'Find and Fix
Cost Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@POUInstalled,@DummyProb,nil,'POU Installed
Count',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@POUInstalledPopulation,@DummyProb,nil,'POU Installed
Population',mtInfo,False,False,False,Config.OptionName,0,True);

  Outputs.AddOutputMetric(@TotalPWSCost,@DummyProb,nil,'_Total PWS
Cost',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalHHCostPWS,@DummyProb,nil,'_Total HH
Cost_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalStateCost,@DummyProb,nil,'_Total State
Cost',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalRuleCost,@DummyProb,nil,'_Total Rule
```

```
Cost',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalCostCap,@DummyProb,nil,'_Total PWS
Cost_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  Outputs.AddOutputMetric(@TotalPWSSamp,@DummyProb,nil,'Total PWS
Sampling',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalPWSSampPWS,@DummyProb,nil,'Total PWS
Sampling_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  Outputs.AddOutputMetric(@TotalPWSLSLR,@DummyProb,nil,'Total PWS
LSLR',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalPWSLSLRMandatory,@DummyProb,nil,'Total PWS LSLR
Mandatory',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalPWSLSLRVoluntary,@DummyProb,nil,'Total PWS LSLR
Voluntary',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalPWSLSLRRequested,@DummyProb,nil,'Total PWS LSLR
Requested',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@LSLReplacementHouseholdTotal,@DummyProb,nil,'LSL
Replacement Household
Total',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  Outputs.AddOutputMetric(@TotalPWSLSLRPWS,@DummyProb,nil,'Total PWS
LSLR_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalPWSLSLRPWSMandatory,@DummyProb,nil,'Total PWS LSLR
Mandatory_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalPWSLSLRPWSVoluntary,@DummyProb,nil,'Total PWS LSLR
Voluntary_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalPWSLSLRPWSRequested,@DummyProb,nil,'Total PWS LSLR
Requested_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalLSLR,@DummyProb,nil,'Total PWS and HH
LSLR',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalLSLRPWS,@DummyProb,nil,'Total PWS and HH
LSLR_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  Outputs.AddOutputMetric(@TotalCCT,@DummyProb,nil,'Total
CCT',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalCCTPWS,@DummyProb,nil,'Total
CCT_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  Outputs.AddOutputMetric(@TotalEDU,@DummyProb,nil,'Total Public
Education',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@TotalEDUPWS,@DummyProb,nil,'Total Public
Education_PWS',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  Outputs.AddOutputMetric(@CostRevenueRatio,@DummyProb,nil,'Cost Revenue
Ratio',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
  Outputs.AddOutputMetric(@CRGT1,@DummyProb,nil,'Cost Revenue Ratio Greater Than
1%',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);
```

```
  Outputs.AddOutputMetric(@CRGT3,@DummyProb,nil,'Cost Revenue Ratio Greater Than
3%',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  for i:=1 to Config.YearsOfAnalysis do begin
    ist:=i.ToString;
    if i<10 then ist:='0'+ist;
    Outputs.AddOutputMetric(@TotalPWSCostUPrivate[i],@DummyProb,nil,'_Total PWS Cost
UMRA Private Y'+ist,mtUMRA,False,False,False,Config.OptionName,0,True);
    Outputs.AddOutputMetric(@TotalPWSCostUPublic[i],@DummyProb,nil,'_Total PWS Cost
UMRA Public Y'+ist,mtUMRA,False,False,False,Config.OptionName,0,True);
    Outputs.AddOutputMetric(@TotalStateCostU[i],@DummyProb,nil,'_Total State Cost
UMRA Y'+ist,mtUMRA,False,False,False,Config.OptionName,0,True);
    Outputs.AddOutputMetric(@TotalLSLHHCostU[i],@DummyProb,nil,'_Total LSL
Replacement HH Cost UMRA Y'+ist,mtUMRA,False,False,False,Config.OptionName,0,True);
  end;

  Outputs.AddOutputMetric(@aTotalPWSCostUPrivate,@DummyProb,nil,'_Total PWS Cost
UCHECK Private',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@aTotalPWSCostUPublic,@DummyProb,nil,'_Total PWS Cost
UCHECK Public',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@aTotalStateCostU,@DummyProb,nil,'_Total State Cost
UCHECK',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@aTotalLSLHHCostU,@DummyProb,nil,'_Total LSL Replacement
HH Cost UCHECK',mtCost,False,False,False,Config.OptionName,0,True);


  i := -1;
  for DictItem in AggregatedResults do
  begin
    Inc(i);
    if not DictItem.Value.IsMain then continue;
    if DictItem.Value.AggType = '2' then
      Outputs.AddOutputMetric(@fResults[i],@DummyProb,
        nil,DictItem.Key,mtCost,False,False,False,
        Config.OptionName,Config.DiscountRate,true)
    else
    if DictItem.Value.AggType = '2PWS' then
      Outputs.AddOutputMetric(@fResultsPWS[i],@DummyProb,
        nil,DictItem.Key,mtCost,False,False,False,
        Config.OptionName,Config.DiscountRate,true);
  end;

  Outputs.AddOutputMetric(@fResultsCapital[0],@DummyProb,
        nil,'System LSLR Capital Cost',mtCost,False,False,False,
        Config.OptionName,Config.DiscountRate,true);

  Outputs.AddOutputMetric(@fResultsCapital[1],@DummyProb,
        nil,'Household LSLR Capital Cost',mtCost,False,False,False,
```

```
        Config.OptionName,Config.DiscountRate,true);

  Outputs.AddOutputMetric(@fResultsCapital[2],@DummyProb,
        nil,'System CCT Capital Cost',mtCost,False,False,False,
        Config.OptionName,Config.DiscountRate,true);

  Outputs.AddOutputMetric(@POTWCost,@DummyProb,nil,'POTW
Cost',mtCost,False,False,False,Config.OptionName,Config.DiscountRate,True);

  Outputs.AddOutputMetric(@prerule_ploading_lbs_5,@DummyProb,nil,'Prerule Ploading
Lbs_5',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@prerule_ploading_lbs_15,@DummyProb,nil,'Prerule Ploading
Lbs_15',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@prerule_ploading_lbs_25,@DummyProb,nil,'Prerule Ploading
Lbs_25',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@prerule_ploading_lbs_35,@DummyProb,nil,'Prerule Ploading
Lbs_35',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@postrule_ploading_lbs_5,@DummyProb,nil,'Postrule Ploading
Lbs_5',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@postrule_ploading_lbs_15,@DummyProb,nil,'Postrule
Ploading Lbs_15',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@postrule_ploading_lbs_25,@DummyProb,nil,'Postrule
Ploading Lbs_25',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@postrule_ploading_lbs_35,@DummyProb,nil,'Postrule
Ploading Lbs_35',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@incr_ploading_lbs_5,@DummyProb,nil,'Increment Ploading
Lbs_5',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@incr_ploading_lbs_15,@DummyProb,nil,'Increment Ploading
Lbs_15',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@incr_ploading_lbs_25,@DummyProb,nil,'Increment Ploading
Lbs_25',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@incr_ploading_lbs_35,@DummyProb,nil,'Increment Ploading
Lbs_35',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@count_incr_ploading_lbs_5,@DummyProb,nil,'Count Increment
Ploading Lbs_5',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@count_incr_ploading_lbs_15,@DummyProb,nil,'Count
Increment Ploading Lbs_15',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@count_incr_ploading_lbs_25,@DummyProb,nil,'Count
Increment Ploading Lbs_25',mtCost,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@count_incr_ploading_lbs_35,@DummyProb,nil,'Count
Increment Ploading Lbs_35',mtCost,False,False,False,Config.OptionName,0,True);

  ResetCosts;

  CCTCostEquations := TCCTCostEquations.Create;
  CCTCostEquations.readSpreadSheet(Config.CCTCostEquations);
  CCTCostEquations.CCTCostEquationLevel := Config.CCTCostEquationLevel;
```

```
  HasCCTCnt := 0;
  NoCCTCnt := 0;
  HasLSLCnt := 0;
  NoLSLCnt := 0;
  SystemAFlowBaseline:=0;
  SystemAFlowOption:=0;

  Flip:=1;
  if Config.RunBaselineOnly then Flip:=-1;

  counter := 0;
end;

destructor TLCRCosts.Destroy;
begin
  CCTCostEquations.Free;
  AggregatedResults.Free;
  BaseCostSteps.Free;
  ScenCostSteps.Free;
  BaseCostVars.Free;
  ScenCostVars.Free;
  inherited;
end;

procedure TLCRCosts.SetCCTCostEquationsData;
begin
  CCTCostEquations.DFlowEP := CostingData.DFlowEP;
  CCTCostEquations.AFlowEP := CostingData.AFlowEP;
  CCTCostEquations.EntryPoints := CostingData.EntryPoints;
  CCTCostEquations.iSystemSize := CostingData.SystemSize;
  CCTCostEquations.iSourceWater := CostingData.SourceWater;

  CCTCostEquations.pbaseph := CostingData.CCTPH;
  CCTCostEquations.pbasepo4 := CostingData.CCTPO4;
  CCTCostEquations.pbasephpo4 := CostingData.CCTBoth;

  CCTCostEquations.iBaselinepo4dose := CostingData.BaselinePO4Dose;
  CCTCostEquations.iBaselineph_wph := CostingData.BaselinePH_wPh;
  CCTCostEquations.iBaselineph_woph := CostingData.BaselinePH_woPh;
  CCTCostEquations.iBaselineph_wocct := CostingData.BaselinePH_woCCT;
  CCTCostEquations.iBaselineph_wpo4ph := CostingData.BaselinePH_wPO4Ph;
end;

procedure TLCRCosts.GenerateCosts(RndSeed:integer; slProxies: TStringList;
SchoolSampData: TSchoolSampDataRec);
var
  i : integer;
  dPWSid: string;
```

```
  DictItem: TPair<string, TAggInputRec>;
  HHConsumption: double;
  BaseHasIt, ScenHasIt: integer;
  prtDebug, VLSystem: boolean;
begin
  PWSCount := 1;
  HHConsumption := 0;

  prtDebug := false;

  dPWSid := CostingData.PWSid;

  if (CostingData.SystemSize = 9) then
    VLSystem := true
  else
    VLSystem := false;

  if CostingData.CCT = 1 then Inc(HasCCTCnt)
  else Inc(NoCCTCnt);
  if CostingData.LSL = 1 then Inc(HasLSLCnt)
  else Inc(NoLSLCnt);

  if Config.RunBaselineOnly or Config.RunDifference then
  begin
    SetCCTCostEquationsData;
    if UserSeeds then RandSeed:=RndSeed;
    BaseCostSteps.SetRandomYears;
    BaseCostSteps.strLeadConcentrations := strLeadConcentrationsB;

BaseCostSteps.SetVariablesAndCalculate(CostingData.SystemSize,CostingData.SourceWate
r,

CostingData.LSL,CostingData.CCT,CostingData.SystemType,CostingData.EntryPoints,
      min(CostingData.Population,CostingData.Connections),
      CostingData.First_ale, CostingData.Population,CostingData.CostCapital, True,
dPWSid, 'Baseline',
      CCTCostEquations, CostingData.NumberLSLs, CostingData.SamplingWeight,
CostingData.fBaseVars,
      prtDebug, VLSystem, BAddCostingData.Small_Correct, BAddCostingData.Bin,
CostingData.P90_base,
      BAddCostingData.Num_Proxies, slProxies, SchoolSampData);

    if VLSystem then
      VLSEpLoop(CostingData, 'Baseline', SchoolSampData);

    for i:=0 to high(BaseCostSteps.AggInputs2) do begin
       AggregatedResults.TryGetValue(BaseCostSteps.AggInputs2[i],AggInput);
       AggInput.BaselineValue := BaseCostSteps.Values2[AggInput.Bidx];
```

```
       AggregatedResults.AddOrSetValue(BaseCostSteps.AggInputs2[i],AggInput);

       AggregatedResults.TryGetValue(Trim(BaseCostSteps.AggInputs2[i]) +
'_PWS',AggInput);
       AggInput.BaselineValue := BaseCostSteps.Values2p[AggInput.Bidxp];
       AggregatedResults.AddOrSetValue(Trim(BaseCostSteps.AggInputs2[i]) +
'_PWS',AggInput);
    end;

    for i:=0 to high(BaseCostSteps.AggICR) do begin
       AggregatedResults.TryGetValue(BaseCostSteps.AggICR[i],AggInput);
       AggInput.BaselineValue := BaseCostSteps.ValuesICR[AggInput.Bidx];
       AggregatedResults.AddOrSetValue(BaseCostSteps.AggICR[i],AggInput);
    end;

    HHConsumption := BaseCostSteps.HHConsumption;
  end;

  if Config.RunOptionOnly or Config.RunDifference then
  begin
    SetCCTCostEquationsData;
    if UserSeeds then RandSeed:=RndSeed;
    ScenCostSteps.strLeadConcentrations := strLeadConcentrationsS;
    ScenCostSteps.SetRandomYears;

ScenCostSteps.SetVariablesAndCalculate(CostingData.SystemSize,CostingData.SourceWate
r,

CostingData.LSL,CostingData.CCT,CostingData.SystemType,CostingData.EntryPoints,
      min(CostingData.Population,CostingData.Connections),
      CostingData.First_ale, CostingData.Population,CostingData.CostCapital, True,
dPWSid, Config.OptionName,
      CCTCostEquations, CostingData.NumberLSLs, CostingData.SamplingWeight,
CostingData.fScenVars,
      prtDebug, VLSystem, SAddCostingData.Small_Correct, SAddCostingData.Bin,
CostingData.P90_base,
      SAddCostingData.Num_Proxies, slProxies, SchoolSampData);

    if VLSystem then
      VLSEpLoop(CostingData, Config.OptionName, SchoolSampData);

      for i:=0 to high(ScenCostSteps.AggInputs2) do begin
         AggregatedResults.TryGetValue(ScenCostSteps.AggInputs2[i],AggInput);
         AggInput.OptionValue := ScenCostSteps.Values2[AggInput.Sidx];
         AggregatedResults.AddOrSetValue(ScenCostSteps.AggInputs2[i],AggInput);

         AggregatedResults.TryGetValue(Trim(ScenCostSteps.AggInputs2[i]) +
'_PWS',AggInput);
```

```
         AggInput.OptionValue := ScenCostSteps.Values2p[AggInput.Sidxp];
         AggregatedResults.AddOrSetValue(Trim(ScenCostSteps.AggInputs2[i]) +
'_PWS',AggInput);
      end;

      for i:=0 to high(ScenCostSteps.AggICR) do begin
         AggregatedResults.TryGetValue(ScenCostSteps.AggICR[i],AggInput);
         AggInput.OptionValue := ScenCostSteps.ValuesICR[AggInput.Sidx];
         AggregatedResults.AddOrSetValue(ScenCostSteps.AggICR[i],AggInput);
      end;

    HHConsumption := ScenCostSteps.HHConsumption;
  end;

  LSLReplacement := 0;
  LSLReplacementMandatory := 0;
  LSLReplacementVoluntary := 0;
  LSLReplacementPopulation := 0;
  LSLReplacementPopMandatory := 0;
  LSLReplacementPopVoluntary := 0;
  LSLReplacementRequested := 0;
  LSLReplacementPopRequested := 0;
  AddModifyCCT := 0;
  AddModifyCCTPopulation := 0;
  LSLReplaced := 0;
  LSLReplacedMandatory := 0;
  LSLReplacedVoluntary := 0;
  LSLReplacedRequested := 0;

  CCTInstalled := 0;
  CCTInstalledPopulation := 0;
  CCTAdjusted := 0;
  CCTAdjusted_ale := 0;
  CCTAdjusted_tle:= 0;
  CCTAdjustedPopulation := 0;
  CCTAdjustedPopulation_ale := 0;
  CCTAdjustedPopulation_tle := 0;
  CCTExisting := 0;
  CCTExistingPopulation := 0;
  FindAndFixCost := 0;
  FindAndFixCostPopulation := 0;
  POUInstalled := 0;
  POUInstalledPopulation := 0;

      prerule_ploading_lbs_5 := 0;
      prerule_ploading_lbs_15 := 0;
      prerule_ploading_lbs_25 := 0;
      prerule_ploading_lbs_35 := 0;
```

```
      postrule_ploading_lbs_5 := 0;
      postrule_ploading_lbs_15 := 0;
      postrule_ploading_lbs_25 := 0;
      postrule_ploading_lbs_35 := 0;
      incr_ploading_lbs_5 := 0;
      incr_ploading_lbs_15 := 0;
      incr_ploading_lbs_25 := 0;
      incr_ploading_lbs_35 := 0;
      count_incr_ploading_lbs_5 := 0;
      count_incr_ploading_lbs_15 := 0;
      count_incr_ploading_lbs_25 := 0;
      count_incr_ploading_lbs_35 := 0;

  if Config.RunBaselineOnly then
  begin
    if not VLSystem then
    begin
      if (BaseCostSteps.LSLReplaced > 0) or (BaseCostSteps.LSLRequested > 0) then
      begin
        LSLReplacement := 1;
        LSLReplacementPopulation :=
round(max(BaseCostSteps.LSLReplaced,BaseCostSteps.LSLRequested) *

(CostingData.Population/CostingData.Connections));
      end;
      if BaseCostSteps.LSLReplacedMandatory > 0 then
      begin
        LSLReplacementMandatory := 1;
        LSLReplacementPopMandatory := round(BaseCostSteps.LSLReplacedMandatory *

(CostingData.Population/CostingData.Connections));
      end;
      if BaseCostSteps.LSLReplacedVoluntary > 0 then
      begin
        LSLReplacementVoluntary := 1;
        LSLReplacementPopVoluntary := round(BaseCostSteps.LSLReplacedVoluntary *

(CostingData.Population/CostingData.Connections));
      end;
      if BaseCostSteps.LSLRequested > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := round(BaseCostSteps.LSLRequested *

(CostingData.Population/CostingData.Connections));
      end;

      if baseCostSteps.HasCCTCost then
```

```
begin
  AddModifyCCT := 1;
  AddModifyCCTPopulation := CostingData.Population;
end;

LSLReplaced := BaseCostSteps.LSLReplaced;
LSLReplacedMandatory := BaseCostSteps.LSLReplacedMandatory;
LSLReplacedVoluntary := BaseCostSteps.LSLReplacedVoluntary;
LSLReplacedRequested := BaseCostSteps.LSLRequested;
LSLReplaced := LSLReplaced + LSLReplacedRequested;

if baseCostSteps.CCTInstalled then
begin
  CCTInstalled := 1;
  CCTInstalledPopulation := CostingData.Population;
end;
if baseCostSteps.CCTAdjusted then
begin
  CCTAdjusted := 1;
  CCTAdjustedPopulation := CostingData.Population;
  CCTAdjusted_ale := 1;
  CCTAdjustedPopulation_ale := CostingData.Population;
end;
if baseCostSteps.CCTExisting then
begin
  CCTExisting := 1;
  CCTExistingPopulation := CostingData.Population;
end;
if baseCostSteps.HasFindAndFixCost then
begin
  FindAndFixCost := 1;
  FindAndFixCostPopulation := CostingData.Population;
end;
if baseCostSteps.POUInstalled then
begin
  POUInstalled := 1;
  POUInstalledPopulation := CostingData.Population;
end;

fResultsCapital[0] := baseCostSteps.ValuesCapital[0];
fResultsCapital[1] := baseCostSteps.ValuesCapital[1];
fResultsCapital[2] := baseCostSteps.ValuesCapital[2];

POTWCost := baseCostSteps.POTWCost;

prerule_ploading_lbs_5 := baseCostSteps.prerule_ploading_lbs_5;
prerule_ploading_lbs_15 := baseCostSteps.prerule_ploading_lbs_15;
prerule_ploading_lbs_25 := baseCostSteps.prerule_ploading_lbs_25;
```

```
      prerule_ploading_lbs_35 := baseCostSteps.prerule_ploading_lbs_35;
      postrule_ploading_lbs_5 := baseCostSteps.postrule_ploading_lbs_5;
      postrule_ploading_lbs_15 := baseCostSteps.postrule_ploading_lbs_15;
      postrule_ploading_lbs_25 := baseCostSteps.postrule_ploading_lbs_25;
      postrule_ploading_lbs_35 := baseCostSteps.postrule_ploading_lbs_35;
      incr_ploading_lbs_5 := baseCostSteps.incr_ploading_lbs_5;
      incr_ploading_lbs_15 := baseCostSteps.incr_ploading_lbs_15;
      incr_ploading_lbs_25 := baseCostSteps.incr_ploading_lbs_25;
      incr_ploading_lbs_35 := baseCostSteps.incr_ploading_lbs_35;
      count_incr_ploading_lbs_5 := baseCostSteps.count_incr_ploading_lbs_5;
      count_incr_ploading_lbs_15 := baseCostSteps.count_incr_ploading_lbs_15;
      count_incr_ploading_lbs_25 := baseCostSteps.count_incr_ploading_lbs_25;
      count_incr_ploading_lbs_35 := baseCostSteps.count_incr_ploading_lbs_35;
    end
    else
    begin
      if (BaseCostSteps.LSLReplacedVLS > 0) or (BaseCostSteps.LSLRequestedVLS > 0)
then
      begin
        LSLReplacement := 1;
        LSLReplacementPopulation :=
max(BaseCostSteps.LSLReplacedPopVLS,BaseCostSteps.LSLRequestedPopVLS);
      end;
      if BaseCostSteps.LSLReplacedMandatoryVLS > 0 then
      begin
        LSLReplacementMandatory := 1;
        LSLReplacementPopMandatory := BaseCostSteps.LSLReplacedPopVLS;
      end;
      if BaseCostSteps.LSLReplacedVoluntaryVLS > 0 then
      begin
        LSLReplacementVoluntary := 1;
        LSLReplacementPopVoluntary := BaseCostSteps.LSLReplacedPopVLS;
      end;
      if BaseCostSteps.LSLRequestedVLS > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := BaseCostSteps.LSLRequestedPopVLS;
      end;

      if baseCostSteps.HasCCTCostVLS then
      begin
        AddModifyCCT := 1;
        AddModifyCCTPopulation := BaseCostSteps.CCTInstalledPopVLS +
BaseCostSteps.CCTAdjustedPopVLS;
      end;

      LSLReplaced := BaseCostSteps.LSLReplacedVLS;
      LSLReplacedMandatory := BaseCostSteps.LSLReplacedMandatoryVLS;
```

```
LSLReplacedVoluntary := BaseCostSteps.LSLReplacedVoluntaryVLS;
LSLReplacedRequested := BaseCostSteps.LSLRequestedVLS;
LSLReplaced := LSLReplaced + LSLReplacedRequested;

if baseCostSteps.CCTInstalledPopVLS > 0 then
begin
  CCTInstalled := 1;
  CCTInstalledPopulation := BaseCostSteps.CCTInstalledPopVLS;
end;
if baseCostSteps.CCTAdjustedPopVLS > 0 then
begin
  CCTAdjusted := 1;
  CCTAdjustedPopulation := BaseCostSteps.CCTAdjustedPopVLS;
  CCTAdjusted_ale := 1;
  CCTAdjustedPopulation_ale := BaseCostSteps.CCTAdjustedPopVLS;
end;

if baseCostSteps.CCTExistingPopVLS > 0 then
begin
  CCTExisting := 1;
  CCTExistingPopulation := BaseCostSteps.CCTExistingPopVLS;
end;
if baseCostSteps.HasFindAndFixCostPopVLS > 0 then
begin
  FindAndFixCost := 1;
  FindAndFixCostPopulation := BaseCostSteps.HasFindAndFixCostPopVLS;
end;
if baseCostSteps.POUInstalledPopVLS > 0 then
begin
  POUInstalled := 1;
  POUInstalledPopulation := BaseCostSteps.POUInstalledPopVLS;
end;

fResultsCapital[0] := baseCostSteps.ValuesCapital[0];
fResultsCapital[1] := baseCostSteps.ValuesCapital[1];
fResultsCapital[2] := baseCostSteps.ValuesCapital[2];

POTWCost := baseCostSteps.POTWCostVLS;

prerule_ploading_lbs_5 := baseCostSteps.prerule_ploading_lbs_5VLS;
prerule_ploading_lbs_15 := baseCostSteps.prerule_ploading_lbs_15VLS;
prerule_ploading_lbs_25 := baseCostSteps.prerule_ploading_lbs_25VLS;
prerule_ploading_lbs_35 := baseCostSteps.prerule_ploading_lbs_35VLS;
postrule_ploading_lbs_5 := baseCostSteps.postrule_ploading_lbs_5VLS;
postrule_ploading_lbs_15 := baseCostSteps.postrule_ploading_lbs_15VLS;
postrule_ploading_lbs_25 := baseCostSteps.postrule_ploading_lbs_25VLS;
postrule_ploading_lbs_35 := baseCostSteps.postrule_ploading_lbs_35VLS;
incr_ploading_lbs_5 := baseCostSteps.incr_ploading_lbs_5VLS;
```

```
      incr_ploading_lbs_15 := baseCostSteps.incr_ploading_lbs_15VLS;
      incr_ploading_lbs_25 := baseCostSteps.incr_ploading_lbs_25VLS;
      incr_ploading_lbs_35 := baseCostSteps.incr_ploading_lbs_35VLS;
      count_incr_ploading_lbs_5 := baseCostSteps.count_incr_ploading_lbs_5VLS;
      count_incr_ploading_lbs_15 := baseCostSteps.count_incr_ploading_lbs_15VLS;
      count_incr_ploading_lbs_25 := baseCostSteps.count_incr_ploading_lbs_25VLS;
      count_incr_ploading_lbs_35 := baseCostSteps.count_incr_ploading_lbs_35VLS;

      SystemAFlowBaseline := baseCostSteps.SystemAFlowVLS;
    end;
  end
  else if Config.RunOptionOnly then
  begin
    if not VLSystem then
    begin
      if (ScenCostSteps.LSLReplaced > 0) or (ScenCostSteps.LSLRequested > 0) then
      begin
        LSLReplacement := 1;
        LSLReplacementPopulation :=
round(max(ScenCostSteps.LSLReplaced,ScenCostSteps.LSLRequested) *

(CostingData.Population/CostingData.Connections));
      end;
      if ScenCostSteps.LSLReplacedMandatory > 0 then
      begin
        LSLReplacementMandatory := 1;
        LSLReplacementPopMandatory := round(ScenCostSteps.LSLReplacedMandatory *

(CostingData.Population/CostingData.Connections));
      end;
      if ScenCostSteps.LSLReplacedVoluntary > 0 then
      begin
        LSLReplacementVoluntary := 1;
        LSLReplacementPopVoluntary := round(ScenCostSteps.LSLReplacedVoluntary *

(CostingData.Population/CostingData.Connections));
      end;
      if ScenCostSteps.LSLRequested > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := round(ScenCostSteps.LSLRequested *

(CostingData.Population/CostingData.Connections));
      end;

      if ScenCostSteps.HasCCTCost then
      begin
        AddModifyCCT := 1;
```

```
  AddModifyCCTPopulation := CostingData.Population;
end;

LSLReplaced := ScenCostSteps.LSLReplaced;
LSLReplacedMandatory := ScenCostSteps.LSLReplacedMandatory;
LSLReplacedVoluntary := ScenCostSteps.LSLReplacedVoluntary;
LSLReplacedRequested := ScenCostSteps.LSLRequested;
LSLReplaced := LSLReplaced + LSLReplacedRequested;

if ScenCostSteps.CCTInstalled then
begin
  CCTInstalled := 1;
  CCTInstalledPopulation := CostingData.Population;
end;
if ScenCostSteps.CCTAdjusted then
begin
  CCTAdjusted := 1;
  CCTAdjustedPopulation := CostingData.Population;

  if ScenCostSteps.CCTAdjusted_ale then
  begin
    CCTAdjusted_ale := 1;
    CCTAdjustedPopulation_ale := CostingData.Population;
  end;

  if ScenCostSteps.CCTAdjusted_tle then
  begin
    CCTAdjusted_tle := 1;
    CCTAdjustedPopulation_tle := CostingData.Population;
  end;
end;
if ScenCostSteps.CCTExisting then
begin
  CCTExisting := 1;
  CCTExistingPopulation := CostingData.Population;
end;
if ScenCostSteps.HasFindAndFixCost then
begin
  FindAndFixCost := 1;
  FindAndFixCostPopulation := CostingData.Population;
end;
if ScenCostSteps.POUInstalled then
begin
  POUInstalled := 1;
  POUInstalledPopulation := CostingData.Population;
end;

fResultsCapital[0] := ScenCostSteps.ValuesCapital[0];
```

```
      fResultsCapital[1] := ScenCostSteps.ValuesCapital[1];
      fResultsCapital[2] := ScenCostSteps.ValuesCapital[2];

      POTWCost := ScenCostSteps.POTWCost;

      prerule_ploading_lbs_5 := ScenCostSteps.prerule_ploading_lbs_5;
      prerule_ploading_lbs_15 := ScenCostSteps.prerule_ploading_lbs_15;
      prerule_ploading_lbs_25 := ScenCostSteps.prerule_ploading_lbs_25;
      prerule_ploading_lbs_35 := ScenCostSteps.prerule_ploading_lbs_35;
      postrule_ploading_lbs_5 := ScenCostSteps.postrule_ploading_lbs_5;
      postrule_ploading_lbs_15 := ScenCostSteps.postrule_ploading_lbs_15;
      postrule_ploading_lbs_25 := ScenCostSteps.postrule_ploading_lbs_25;
      postrule_ploading_lbs_35 := ScenCostSteps.postrule_ploading_lbs_35;
      incr_ploading_lbs_5 := ScenCostSteps.incr_ploading_lbs_5;
      incr_ploading_lbs_15 := ScenCostSteps.incr_ploading_lbs_15;
      incr_ploading_lbs_25 := ScenCostSteps.incr_ploading_lbs_25;
      incr_ploading_lbs_35 := ScenCostSteps.incr_ploading_lbs_35;
      count_incr_ploading_lbs_5 := ScenCostSteps.count_incr_ploading_lbs_5;
      count_incr_ploading_lbs_15 := ScenCostSteps.count_incr_ploading_lbs_15;
      count_incr_ploading_lbs_25 := ScenCostSteps.count_incr_ploading_lbs_25;
      count_incr_ploading_lbs_35 := ScenCostSteps.count_incr_ploading_lbs_35;
    end
    else // VLSystem
    begin
      if (ScenCostSteps.LSLReplacedVLS > 0) or (ScenCostSteps.LSLRequestedVLS > 0)
then
      begin
        LSLReplacement := 1;
        LSLReplacementPopulation :=
max(ScenCostSteps.LSLReplacedPopVLS,ScenCostSteps.LSLRequestedPopVLS);
      end;
      if ScenCostSteps.LSLReplacedMandatoryVLS > 0 then
      begin
        LSLReplacementMandatory := 1;
        LSLReplacementPopMandatory := ScenCostSteps.LSLReplacedPopVLS;
      end;
      if ScenCostSteps.LSLReplacedVoluntaryVLS > 0 then
      begin
        LSLReplacementVoluntary := 1;
        LSLReplacementPopVoluntary := ScenCostSteps.LSLReplacedPopVLS;
      end;
      if ScenCostSteps.LSLRequestedVLS > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := ScenCostSteps.LSLRequestedPopVLS;
      end;

      if ScenCostSteps.HasCCTCostVLS then
```

```
      begin
        AddModifyCCT := 1;
        AddModifyCCTPopulation := ScenCostSteps.CCTInstalledPopVLS +
ScenCostSteps.CCTAdjustedPopVLS;
      end;

      LSLReplaced := ScenCostSteps.LSLReplacedVLS;
      LSLReplacedMandatory := ScenCostSteps.LSLReplacedMandatoryVLS;
      LSLReplacedVoluntary := ScenCostSteps.LSLReplacedVoluntaryVLS;
      LSLReplacedRequested := ScenCostSteps.LSLRequestedVLS;
      LSLReplaced := LSLReplaced + LSLReplacedRequested;

      if ScenCostSteps.CCTInstalledPopVLS > 0 then
      begin
        CCTInstalled := 1;
        CCTInstalledPopulation := ScenCostSteps.CCTInstalledPopVLS;
      end;

      if ScenCostSteps.CCTAdjustedPopVLS > 0 then
      begin
        CCTAdjusted := 1;
        CCTAdjustedPopulation := ScenCostSteps.CCTAdjustedPopVLS;

        if ScenCostSteps.CCTAdjustedPopVLS_ale > 0 then
        begin
          CCTAdjusted_ale := 1;
          CCTAdjustedPopulation_ale := ScenCostSteps.CCTAdjustedPopVLS_ale;
        end;

        if ScenCostSteps.CCTAdjustedPopVLS_tle > 0 then
        begin
          CCTAdjusted_tle := 1;
          CCTAdjustedPopulation_tle := ScenCostSteps.CCTAdjustedPopVLS_tle;
        end;
      end;

      if ScenCostSteps.CCTExistingPopVLS > 0 then
      begin
        CCTExisting := 1;
        CCTExistingPopulation := ScenCostSteps.CCTExistingPopVLS;
      end;
      if ScenCostSteps.HasFindAndFixCostPopVLS > 0 then
      begin
        FindAndFixCost := 1;
        FindAndFixCostPopulation := ScenCostSteps.HasFindAndFixCostPopVLS;
      end;
      if ScenCostSteps.POUInstalledPopVLS > 0 then
      begin
```

```
        POUInstalled := 1;
        POUInstalledPopulation := ScenCostSteps.POUInstalledPopVLS;
      end;

      fResultsCapital[0] := ScenCostSteps.ValuesCapital[0];
      fResultsCapital[1] := ScenCostSteps.ValuesCapital[1];
      fResultsCapital[2] := ScenCostSteps.ValuesCapital[2];

      POTWCost := ScenCostSteps.POTWCostVLS;

      prerule_ploading_lbs_5 := ScenCostSteps.prerule_ploading_lbs_5VLS;
      prerule_ploading_lbs_15 := ScenCostSteps.prerule_ploading_lbs_15VLS;
      prerule_ploading_lbs_25 := ScenCostSteps.prerule_ploading_lbs_25VLS;
      prerule_ploading_lbs_35 := ScenCostSteps.prerule_ploading_lbs_35VLS;
      postrule_ploading_lbs_5 := ScenCostSteps.postrule_ploading_lbs_5VLS;
      postrule_ploading_lbs_15 := ScenCostSteps.postrule_ploading_lbs_15VLS;
      postrule_ploading_lbs_25 := ScenCostSteps.postrule_ploading_lbs_25VLS;
      postrule_ploading_lbs_35 := ScenCostSteps.postrule_ploading_lbs_35VLS;
      incr_ploading_lbs_5 := ScenCostSteps.incr_ploading_lbs_5VLS;
      incr_ploading_lbs_15 := ScenCostSteps.incr_ploading_lbs_15VLS;
      incr_ploading_lbs_25 := ScenCostSteps.incr_ploading_lbs_25VLS;
      incr_ploading_lbs_35 := ScenCostSteps.incr_ploading_lbs_35VLS;
      count_incr_ploading_lbs_5 := ScenCostSteps.count_incr_ploading_lbs_5VLS;
      count_incr_ploading_lbs_15 := ScenCostSteps.count_incr_ploading_lbs_15VLS;
      count_incr_ploading_lbs_25 := ScenCostSteps.count_incr_ploading_lbs_25VLS;
      count_incr_ploading_lbs_35 := ScenCostSteps.count_incr_ploading_lbs_35VLS;

      SystemAFlowOption := scenCostSteps.SystemAFlowVLS;
    end;

  end
  else if Config.RunDifference then
  begin
    if not VLSystem then
    begin
      BaseHasIt := 0;
      ScenHasIt := 0;
      if (BaseCostSteps.LSLReplaced > 0) or (BaseCostSteps.LSLRequested > 0) then
BaseHasIt := 1;
      if (ScenCostSteps.LSLReplaced > 0) or (ScenCostSteps.LSLRequested > 0) then
ScenHasIt := 1;
      LSLReplacement := ScenHasIt - BaseHasIt;
      if (ScenCostSteps.LSLReplaced > 0) or (ScenCostSteps.LSLRequested > 0) then
        LSLReplacementPopulation :=
round(max(ScenCostSteps.LSLReplaced,ScenCostSteps.LSLRequested) *

(CostingData.Population/CostingData.Connections));
      if (BaseCostSteps.LSLReplaced > 0) or (BaseCostSteps.LSLRequested > 0)then
```

```
        LSLReplacementPopulation := LSLReplacementPopulation -
round(max(BaseCostSteps.LSLReplaced,BaseCostSteps.LSLRequested) *

(CostingData.Population/CostingData.Connections));

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.LSLReplacedMandatory > 0 then BaseHasIt := 1;
      if ScenCostSteps.LSLReplacedMandatory > 0 then ScenHasIt := 1;
      LSLReplacementMandatory := ScenHasIt - BaseHasIt;
      if ScenCostSteps.LSLReplacedMandatory > 0 then
        LSLReplacementPopMandatory := round(ScenCostSteps.LSLReplacedMandatory *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.LSLReplacedMandatory > 0 then
        LSLReplacementPopMandatory := LSLReplacementPopMandatory -
round(BaseCostSteps.LSLReplacedMandatory *

(CostingData.Population/CostingData.Connections));

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.LSLReplacedVoluntary > 0 then BaseHasIt := 1;
      if ScenCostSteps.LSLReplacedVoluntary > 0 then ScenHasIt := 1;
      LSLReplacementVoluntary := ScenHasIt - BaseHasIt;
      if ScenCostSteps.LSLReplacedVoluntary > 0 then
        LSLReplacementPopVoluntary := round(ScenCostSteps.LSLReplacedVoluntary *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.LSLReplacedVoluntary > 0 then
        LSLReplacementPopVoluntary := LSLReplacementPopVoluntary -
round(BaseCostSteps.LSLReplacedVoluntary *

(CostingData.Population/CostingData.Connections));

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.LSLRequested > 0 then BaseHasIt := 1;
      if ScenCostSteps.LSLRequested > 0 then ScenHasIt := 1;
      LSLReplacementRequested := ScenHasIt - BaseHasIt;
      if ScenCostSteps.LSLRequested > 0 then
        LSLReplacementPopRequested := round(ScenCostSteps.LSLRequested *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.LSLRequested > 0 then
        LSLReplacementPopRequested := LSLReplacementPopRequested -
round(BaseCostSteps.LSLRequested *
```

```
(CostingData.Population/CostingData.Connections));

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.HasCCTCost then BaseHasIt := 1;
      if ScenCostSteps.HasCCTCost then ScenHasIt := 1;
      AddModifyCCT := ScenHasIt - BaseHasIt;
      AddModifyCCTPopulation := CostingData.Population * AddModifyCCT;

      LSLReplaced := (ScenCostSteps.LSLReplaced + ScenCostSteps.LSLRequested) -
(BaseCostSteps.LSLReplaced + BaseCostSteps.LSLRequested);
      LSLReplacedMandatory := ScenCostSteps.LSLReplacedMandatory -
BaseCostSteps.LSLReplacedMandatory;
      LSLReplacedVoluntary := ScenCostSteps.LSLReplacedVoluntary -
BaseCostSteps.LSLReplacedVoluntary;
      LSLReplacedRequested := ScenCostSteps.LSLRequested -
BaseCostSteps.LSLRequested;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTInstalled then BaseHasIt := 1;
      if ScenCostSteps.CCTInstalled then ScenHasIt := 1;
      CCTInstalled := ScenHasIt - BaseHasIt;
      CCTInstalledPopulation := CostingData.Population * CCTInstalled;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTAdjusted then BaseHasIt := 1;
      if ScenCostSteps.CCTAdjusted then ScenHasIt := 1;
      CCTAdjusted := ScenHasIt - BaseHasIt;
      CCTAdjustedPopulation := CostingData.Population * CCTAdjusted;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTAdjusted_ale then BaseHasIt := 1;
      if ScenCostSteps.CCTAdjusted_ale then ScenHasIt := 1;
      CCTAdjusted_ale := ScenHasIt - BaseHasIt;
      CCTAdjustedPopulation_ale := CostingData.Population * CCTAdjusted_ale;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTAdjusted_tle then BaseHasIt := 1;
      if ScenCostSteps.CCTAdjusted_tle then ScenHasIt := 1;
      CCTAdjusted_tle := ScenHasIt - BaseHasIt;
      CCTAdjustedPopulation_tle := CostingData.Population * CCTAdjusted_tle;

      BaseHasIt := 0;
      ScenHasIt := 0;
```

```
      if BaseCostSteps.CCTExisting then BaseHasIt := 1;
      if ScenCostSteps.CCTExisting then ScenHasIt := 1;
      CCTExisting := ScenHasIt - BaseHasIt;
      CCTExistingPopulation := CostingData.Population * CCTExisting;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.HasFindAndFixCost then BaseHasIt := 1;
      if ScenCostSteps.HasFindAndFixCost then ScenHasIt := 1;
      FindAndFixCost := ScenHasIt - BaseHasIt;
      FindAndFixCostPopulation := CostingData.Population * FindAndFixCost;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.POUInstalled then BaseHasIt := 1;
      if ScenCostSteps.POUInstalled then ScenHasIt := 1;
      POUInstalled := ScenHasIt - BaseHasIt;
      POUInstalledPopulation := CostingData.Population * POUInstalled;

      fResultsCapital[0] := ScenCostSteps.ValuesCapital[0] -
baseCostSteps.ValuesCapital[0];
      fResultsCapital[1] := ScenCostSteps.ValuesCapital[1] -
baseCostSteps.ValuesCapital[1];
      fResultsCapital[2] := ScenCostSteps.ValuesCapital[2] -
baseCostSteps.ValuesCapital[2];

      POTWCost := ScenCostSteps.POTWCost - baseCostSteps.POTWCost;

      SystemAFlowBaseline := baseCostSteps.SystemAFlowVLS;
      SystemAFlowOption := scenCostSteps.SystemAFlowVLS;

      prerule_ploading_lbs_5 := ScenCostSteps.prerule_ploading_lbs_5 -
baseCostSteps.prerule_ploading_lbs_5;
      prerule_ploading_lbs_15 := ScenCostSteps.prerule_ploading_lbs_15 -
baseCostSteps.prerule_ploading_lbs_15;
      prerule_ploading_lbs_25 := ScenCostSteps.prerule_ploading_lbs_25 -
baseCostSteps.prerule_ploading_lbs_25;
      prerule_ploading_lbs_35 := ScenCostSteps.prerule_ploading_lbs_35 -
baseCostSteps.prerule_ploading_lbs_35;
      postrule_ploading_lbs_5 := ScenCostSteps.postrule_ploading_lbs_5 -
baseCostSteps.postrule_ploading_lbs_5;
      postrule_ploading_lbs_15 := ScenCostSteps.postrule_ploading_lbs_15 -
baseCostSteps.postrule_ploading_lbs_15;
      postrule_ploading_lbs_25 := ScenCostSteps.postrule_ploading_lbs_25 -
baseCostSteps.postrule_ploading_lbs_25;
      postrule_ploading_lbs_35 := ScenCostSteps.postrule_ploading_lbs_35 -
baseCostSteps.postrule_ploading_lbs_35;
      incr_ploading_lbs_5 := ScenCostSteps.incr_ploading_lbs_5 -
```

```
baseCostSteps.incr_ploading_lbs_5;
      incr_ploading_lbs_15 := ScenCostSteps.incr_ploading_lbs_15 -
baseCostSteps.incr_ploading_lbs_15;
      incr_ploading_lbs_25 := ScenCostSteps.incr_ploading_lbs_25 -
baseCostSteps.incr_ploading_lbs_25;
      incr_ploading_lbs_35 := ScenCostSteps.incr_ploading_lbs_35 -
baseCostSteps.incr_ploading_lbs_35;

      count_incr_ploading_lbs_5 := ScenCostSteps.count_incr_ploading_lbs_5 -
baseCostSteps.count_incr_ploading_lbs_5;
      count_incr_ploading_lbs_15 := ScenCostSteps.count_incr_ploading_lbs_15 -
baseCostSteps.count_incr_ploading_lbs_15;
      count_incr_ploading_lbs_25 := ScenCostSteps.count_incr_ploading_lbs_25 -
baseCostSteps.count_incr_ploading_lbs_25;
      count_incr_ploading_lbs_35 := ScenCostSteps.count_incr_ploading_lbs_35 -
baseCostSteps.count_incr_ploading_lbs_35;
    end
    else
    begin
      BaseHasIt := 0;
      ScenHasIt := 0;
      if (BaseCostSteps.LSLReplacedVLS > 0) or (BaseCostSteps.LSLRequestedVLS > 0)
then BaseHasIt := 1;
      if (ScenCostSteps.LSLReplacedVLS > 0) or (ScenCostSteps.LSLRequestedVLS > 0)
then ScenHasIt := 1;
      LSLReplacement := ScenHasIt - BaseHasIt;
      LSLReplacementPopulation :=
max(ScenCostSteps.LSLReplacedPopVLS,ScenCostSteps.LSLRequestedPopVLS) -
max(BaseCostSteps.LSLReplacedPopVLS,BaseCostSteps.LSLRequestedPopVLS);

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.LSLReplacedMandatoryVLS > 0 then BaseHasIt := 1;
      if ScenCostSteps.LSLReplacedMandatoryVLS > 0 then ScenHasIt := 1;
      LSLReplacementMandatory := ScenHasIt - BaseHasIt;
      LSLReplacementPopMandatory := (ScenCostSteps.LSLReplacedPopMandatoryVLS -
BaseCostSteps.LSLReplacedPopMandatoryVLS);

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.LSLReplacedVoluntaryVLS > 0 then BaseHasIt := 1;
      if ScenCostSteps.LSLReplacedVoluntaryVLS > 0 then ScenHasIt := 1;
      LSLReplacementVoluntary := ScenHasIt - BaseHasIt;
      LSLReplacementPopVoluntary := (ScenCostSteps.LSLReplacedPopVoluntaryVLS -
BaseCostSteps.LSLReplacedPopVoluntaryVLS);

      BaseHasIt := 0;
      ScenHasIt := 0;
```

```
      if BaseCostSteps.LSLRequestedVLS > 0 then BaseHasIt := 1;
      if ScenCostSteps.LSLRequestedVLS > 0 then ScenHasIt := 1;
      LSLReplacementRequested := ScenHasIt - BaseHasIt;
      LSLReplacementPopRequested := (ScenCostSteps.LSLRequestedPopVLS -
BaseCostSteps.LSLRequestedPopVLS);

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.HasCCTCostVLS then BaseHasIt := 1;
      if ScenCostSteps.HasCCTCostVLS then ScenHasIt := 1;
      AddModifyCCT := ScenHasIt - BaseHasIt;
      AddModifyCCTPopulation := ((ScenCostSteps.CCTInstalledPopVLS +
ScenCostSteps.CCTAdjustedPopVLS) -
                                 (BaseCostSteps.CCTInstalledPopVLS +
BaseCostSteps.CCTAdjustedPopVLS)) * AddModifyCCT;

      LSLReplaced := (ScenCostSteps.LSLReplacedVLS + ScenCostSteps.LSLRequestedVLS)
- (BaseCostSteps.LSLReplacedVLS + BaseCostSteps.LSLRequestedVLS);
      LSLReplacedMandatory := ScenCostSteps.LSLReplacedMandatoryVLS -
BaseCostSteps.LSLReplacedMandatoryVLS;
      LSLReplacedVoluntary := ScenCostSteps.LSLReplacedVoluntaryVLS -
BaseCostSteps.LSLReplacedVoluntaryVLS;
      LSLReplacedRequested := ScenCostSteps.LSLRequestedVLS -
BaseCostSteps.LSLRequestedVLS;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTInstalledPopVLS>0 then BaseHasIt := 1;
      if ScenCostSteps.CCTInstalledPopVLS>0 then ScenHasIt := 1;
      CCTInstalled := ScenHasIt - BaseHasIt;
      CCTInstalledPopulation := (ScenCostSteps.CCTInstalledPopVLS -
BaseCostSteps.CCTInstalledPopVLS) * CCTInstalled;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTAdjustedPopVLS>0 then BaseHasIt := 1;
      if ScenCostSteps.CCTAdjustedPopVLS>0 then ScenHasIt := 1;
      CCTAdjusted := ScenHasIt - BaseHasIt;
      CCTAdjustedPopulation := (ScenCostSteps.CCTAdjustedPopVLS -
BaseCostSteps.CCTAdjustedPopVLS) * CCTAdjusted;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTAdjustedPopVLS_ale>0 then BaseHasIt := 1;
      if ScenCostSteps.CCTAdjustedPopVLS_ale>0 then ScenHasIt := 1;
      CCTAdjusted := ScenHasIt - BaseHasIt;
      CCTAdjustedPopulation := (ScenCostSteps.CCTAdjustedPopVLS_ale -
BaseCostSteps.CCTAdjustedPopVLS_ale) * CCTAdjusted;
```

```
      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.CCTAdjustedPopVLS_tle>0 then BaseHasIt := 1;
      if ScenCostSteps.CCTAdjustedPopVLS_tle>0 then ScenHasIt := 1;
      CCTAdjusted := ScenHasIt - BaseHasIt;
      CCTAdjustedPopulation := (ScenCostSteps.CCTAdjustedPopVLS_tle -
BaseCostSteps.CCTAdjustedPopVLS_ale) * CCTAdjusted;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if BaseCostSteps.HasFindAndFixCostPopVLS>0 then BaseHasIt := 1;
      if ScenCostSteps.HasFindAndFixCostPopVLS>0 then ScenHasIt := 1;
      FindAndFixCost := ScenHasIt - BaseHasIt;
      FindAndFixCostPopulation := (ScenCostSteps.HasFindAndFixCostPopVLS -
BaseCostSteps.HasFindAndFixCostPopVLS);

      fResultsCapital[0] := ScenCostSteps.ValuesCapital[0] -
baseCostSteps.ValuesCapital[0];
      fResultsCapital[1] := ScenCostSteps.ValuesCapital[1] -
baseCostSteps.ValuesCapital[1];
      fResultsCapital[2] := ScenCostSteps.ValuesCapital[2] -
baseCostSteps.ValuesCapital[2];

      POTWCost := ScenCostSteps.POTWCostVLS - baseCostSteps.POTWCostVLS;

      prerule_ploading_lbs_5 := ScenCostSteps.prerule_ploading_lbs_5VLS -
baseCostSteps.prerule_ploading_lbs_5VLS;
      prerule_ploading_lbs_15 := ScenCostSteps.prerule_ploading_lbs_15VLS -
baseCostSteps.prerule_ploading_lbs_15VLS;
      prerule_ploading_lbs_25 := ScenCostSteps.prerule_ploading_lbs_25VLS -
baseCostSteps.prerule_ploading_lbs_25VLS;
      prerule_ploading_lbs_35 := ScenCostSteps.prerule_ploading_lbs_35VLS -
baseCostSteps.prerule_ploading_lbs_35VLS;
      postrule_ploading_lbs_5 := ScenCostSteps.postrule_ploading_lbs_5VLS -
baseCostSteps.postrule_ploading_lbs_5VLS;
      postrule_ploading_lbs_15 := ScenCostSteps.postrule_ploading_lbs_15VLS -
baseCostSteps.postrule_ploading_lbs_15VLS;
      postrule_ploading_lbs_25 := ScenCostSteps.postrule_ploading_lbs_25VLS -
baseCostSteps.postrule_ploading_lbs_25VLS;
      postrule_ploading_lbs_35 := ScenCostSteps.postrule_ploading_lbs_35VLS -
baseCostSteps.postrule_ploading_lbs_35VLS;
      incr_ploading_lbs_5 := ScenCostSteps.incr_ploading_lbs_5VLS -
baseCostSteps.incr_ploading_lbs_5VLS;
      incr_ploading_lbs_15 := ScenCostSteps.incr_ploading_lbs_15VLS -
baseCostSteps.incr_ploading_lbs_15VLS;
      incr_ploading_lbs_25 := ScenCostSteps.incr_ploading_lbs_25VLS -
baseCostSteps.incr_ploading_lbs_25VLS;
```

```
      incr_ploading_lbs_35 := ScenCostSteps.incr_ploading_lbs_35VLS -
baseCostSteps.incr_ploading_lbs_35VLS;

      count_incr_ploading_lbs_5 := ScenCostSteps.count_incr_ploading_lbs_5VLS -
baseCostSteps.count_incr_ploading_lbs_5VLS;
      count_incr_ploading_lbs_15 := ScenCostSteps.count_incr_ploading_lbs_15VLS -
baseCostSteps.count_incr_ploading_lbs_15VLS;
      count_incr_ploading_lbs_25 := ScenCostSteps.count_incr_ploading_lbs_25VLS -
baseCostSteps.count_incr_ploading_lbs_25VLS;
      count_incr_ploading_lbs_35 := ScenCostSteps.count_incr_ploading_lbs_35VLS -
baseCostSteps.count_incr_ploading_lbs_35VLS;

      SystemAFlowBaseline := BaseCostSteps.SystemAFlowVLS;
      SystemAFlowOption := ScenCostSteps.SystemAFlowVLS;
    end;
  end;


  i := -1;
  for DictItem in AggregatedResults do
  begin
    Inc(i);
    if not DictItem.Value.IsMain then continue;

    if Config.RunBaselineOnly then
    begin
      if DictItem.Value.AggType = '2' then
        fResults[i] := DictItem.Value.BaselineValue
      else
      if DictItem.Value.AggType = '2PWS' then
        fResultsPWS[i] := DictItem.Value.BaselineValue
      else
      if DictItem.Value.AggType = 'ICR' then
        fResultsICR[i] := DictItem.Value.BaselineValue;
    end
    else
    if Config.RunOptionOnly then
    begin
      if DictItem.Value.AggType = '2' then
        fResults[i] := DictItem.Value.OptionValue
      else
      if DictItem.Value.AggType = '2PWS' then
        fResultsPWS[i] := DictItem.Value.OptionValue
      else
      if DictItem.Value.AggType = 'ICR' then
        fResultsICR[i] := DictItem.Value.OptionValue;
    end
    else
```

```
    begin
      if DictItem.Value.AggType = '2' then
        fResults[i] := DictItem.Value.OptionValue - DictItem.Value.BaselineValue
      else
      if DictItem.Value.AggType = '2PWS' then
        fResultsPWS[i] := DictItem.Value.OptionValue - DictItem.Value.BaselineValue
      else
      if DictItem.Value.AggType = 'ICR' then
        fResultsICR[i] := DictItem.Value.OptionValue - DictItem.Value.BaselineValue;
    end;
  end;

  CalculateCustomMetrics(HHConsumption, VLSystem);
end;

function TLCRCosts.GetTotalEvaluations: int64;
var t : TCostingStep;
begin
  Result:=0;
  for t in BaseCostSteps.CostSteps do
    Result:=Result + T.EvalCount;
  for t in ScenCostSteps.CostSteps do
    Result:=Result + T.EvalCount;
end;

function TLCRCosts.GetTotalCompiledEvaluations: int64;
var t : TCostingStep;
begin
  Result:=BaseCostSteps.CC.TotEval + ScenCostSteps.CC.TotEval;
end;


procedure TLCRCosts.ResetCosts;
begin
  fillchar(fResults,SizeOf(fResults),0);
  fillchar(fResultsPWS,SizeOf(fResultsPWS),0);
end;

procedure TLCRCosts.StateCosts;
var
  StateCosts, StateICR_C1, StateICR_C2, StateICR_C3, StateICR_C4, StateICR_C10:
double;
  StateICR_H1, StateICR_H2, StateICR_H3, StateICR_H4, StateICR_H10: double;
  SL: TStringlist;
  sLine: string;
begin
  StateCosts := 0;
  StateICR_C1 := 0;
```

```
StateICR_C2 := 0;
StateICR_C3 := 0;
StateICR_C4 := 0;
StateICR_C10 := 0;
StateICR_H1 := 0;
StateICR_H2 := 0;
StateICR_H3 := 0;
StateICR_H4 := 0;
StateICR_H10 := 0;

SL := TStringlist.Create;

if Config.RunBaselineOnly then
begin
  BaseCostSteps.StateCostsCalculate(True, 'Baseline');
  StateCosts := StateCosts + BaseCostSteps.StateCost;

  StateICR_H1 := StateICR_H1 + BaseCostSteps.StateICRHours1;
  StateICR_H2 := StateICR_H2 + BaseCostSteps.StateICRHours2;
  StateICR_H3 := StateICR_H3 + BaseCostSteps.StateICRHours3;
  StateICR_H4 := StateICR_H4 + BaseCostSteps.StateICRHours4;
  StateICR_H10 := StateICR_H10 + BaseCostSteps.StateICRHours10;

  StateICR_C1 := StateICR_C1 + BaseCostSteps.StateICRCost1;
  StateICR_C2 := StateICR_C2 + BaseCostSteps.StateICRCost2;
  StateICR_C3 := StateICR_C3 + BaseCostSteps.StateICRCost3;
  StateICR_C4 := StateICR_C4 + BaseCostSteps.StateICRCost4;
  StateICR_C10 := StateICR_C10 + BaseCostSteps.StateICRCost10;
end
else
if Config.RunOptionOnly then
begin
  ScenCostSteps.StateCostsCalculate(True, 'Option');
  StateCosts := StateCosts + ScenCostSteps.StateCost;

  StateICR_H1 := StateICR_H1 + ScenCostSteps.StateICRHours1;
  StateICR_H2 := StateICR_H2 + ScenCostSteps.StateICRHours2;
  StateICR_H3 := StateICR_H3 + ScenCostSteps.StateICRHours3;
  StateICR_H4 := StateICR_H4 + ScenCostSteps.StateICRHours4;
  StateICR_H10 := StateICR_H10 + ScenCostSteps.StateICRHours10;

  StateICR_C1 := StateICR_C1 + ScenCostSteps.StateICRCost1;
  StateICR_C2 := StateICR_C2 + ScenCostSteps.StateICRCost2;
  StateICR_C3 := StateICR_C3 + ScenCostSteps.StateICRCost3;
  StateICR_C4 := StateICR_C4 + ScenCostSteps.StateICRCost4;
  StateICR_C10 := StateICR_C10 + ScenCostSteps.StateICRCost10;
end
else
```

```
  begin
    BaseCostSteps.StateCostsCalculate(True, 'Baseline');
    ScenCostSteps.StateCostsCalculate(True, 'Option');
    StateCosts := StateCosts + (ScenCostSteps.StateCost - BaseCostSteps.StateCost);

    StateICR_C1 := StateICR_C1 + (ScenCostSteps.StateICRCost1 -
BaseCostSteps.StateICRCost1);
    StateICR_C2 := StateICR_C2 + (ScenCostSteps.StateICRCost2 -
BaseCostSteps.StateICRCost2);
    StateICR_C3 := StateICR_C3 + (ScenCostSteps.StateICRCost3 -
BaseCostSteps.StateICRCost3);
    StateICR_C4 := StateICR_C4 + (ScenCostSteps.StateICRCost4 -
BaseCostSteps.StateICRCost4);
    StateICR_C10 := StateICR_C10 + (ScenCostSteps.StateICRCost10 -
BaseCostSteps.StateICRCost10);

    StateICR_H1 := StateICR_H1 + (ScenCostSteps.StateICRHours1 -
BaseCostSteps.StateICRHours1);
    StateICR_H2 := StateICR_H2 + (ScenCostSteps.StateICRHours2 -
BaseCostSteps.StateICRHours2);
    StateICR_H3 := StateICR_H3 + (ScenCostSteps.StateICRHours3 -
BaseCostSteps.StateICRHours3);
    StateICR_H4 := StateICR_H4 + (ScenCostSteps.StateICRHours4 -
BaseCostSteps.StateICRHours4);
    StateICR_H10 := StateICR_H10 + (ScenCostSteps.StateICRHours10 -
BaseCostSteps.StateICRHours10);
  end;

  sLine := 'State Cost'+chr(9)+'State Cost 1'+chr(9)+'State Cost 2'+chr(9)+'State
Cost 3'+chr(9)+'State Cost 4'+chr(9)+'State Cost 10';
  sLine := sLine+chr(9)+'State Hours 1'+chr(9)+'State Hours 2'+chr(9)+'State Hours
3'+chr(9)+'State Hours 4'+chr(9)+'State Hours 10';
  SL.Add(sLine);
  sLine := StateCosts.ToString + chr(9) + StateICR_C1.ToString + chr(9) +
StateICR_C2.ToString + chr(9) +
           StateICR_C3.ToString + chr(9) + StateICR_C4.ToString + chr(9) +
StateICR_C10.ToString;
  sLine := sLine + chr(9) + StateICR_H1.ToString + chr(9) + StateICR_H2.ToString +
chr(9) +
           StateICR_H3.ToString + chr(9) + StateICR_H4.ToString + chr(9) +
StateICR_H10.ToString;
  SL.Add(sLIne);
  SL.SaveToFile(UserPath + Config.RunName + '_StateCosts.tab');
  SL.Free;
end;

procedure TLCRCosts.VLSEpLoop(CostingData: TCostGenRec; option: string;
SchoolSampData: TSchoolSampDataRec);
```

```
var
  Xls: TExcelFile;
  r: integer;
  idcnt:integer;
  Filename: string;

  prtDebug: boolean;
  AFlow, DFlow: double;
  iNumEpEntries: integer;
  iNumLSLEps: integer;

  function NumEpEntries: integer;
  var
    r, cnt: integer;
  begin
    Xls := TXlsFile.Create(Filename, False);
    Xls.ActiveSheetByName := 'Inputs';
    cnt := 0;
    iNumLSLEps := 0;

    for r := 2 to Xls.RowCount do
    begin
      if Xls.GetStringFromCell(r,1) = Copy(CostingData.PWSId,1,9) then
      begin
        cnt := cnt + 1;
        if Xls.GetStringFromCell(r,10) = '1' then
          iNumLSLEps := iNumLSLEps + 1;
      end;
    end;
    FreeAndNil(Xls);
    Result := cnt;
  end;
begin
  idcnt:=0;
  prtDebug := false;

  Config.GetFlows(CostingData.Ownership, CostingData.SourceWater,
CostingData.Population,
                  AFlow, DFlow);

  if option = 'Baseline' then
  begin
    BaseCostSteps.HasLSLRCostVLS := false;
    BaseCostSteps.HasCCTCostVLS := false;
    BaseCostSteps.LSLReplacedVLS := 0;
    BaseCostSteps.LSLReplacedPopVLS := 0;
    BaseCostSteps.LSLReplacedMandatoryVLS := 0;
    BaseCostSteps.LSLReplacedVoluntaryVLS := 0;
```

```
  BaseCostSteps.LSLReplacedPopMandatoryVLS := 0;
  BaseCostSteps.LSLReplacedPopVoluntaryVLS := 0;
  BaseCostSteps.LSLRequestedVLS := 0;
  BaseCostSteps.LSLRequestedPopVLS := 0;
  BaseCostSteps.CCTInstalledVLS := false;
  BaseCostSteps.CCTAdjustedVLS := false;
  BaseCostSteps.CCTAdjustedVLS_ale := false;
  BaseCostSteps.CCTAdjustedVLS_tle := false;
  BaseCostSteps.CCTInstalledPopVLS := 0;
  BaseCostSteps.CCTAdjustedPopVLS := 0;
  BaseCostSteps.CCTAdjustedPopVLS_ale := 0;
  BaseCostSteps.CCTAdjustedPopVLS_tle := 0;
  BaseCostSteps.SystemAFlowVLS := AFlow;
  BaseCostSteps.CCTExistingVLS := false;
  BaseCostSteps.HasFindAndFixCostVLS := false;
  BaseCostSteps.POUInstalledVLS := false;
  BaseCostSteps.CCTExistingPopVLS := 0;
  BaseCostSteps.HasFindAndFixCostPopVLS := 0;
  BaseCostSteps.POUInstalledPopVLS := 0;

  BaseCostSteps.POTWCostVLS := 0;

  BaseCostSteps.prerule_ploading_lbs_5VLS := 0;
  BaseCostSteps.prerule_ploading_lbs_15VLS := 0;
  BaseCostSteps.prerule_ploading_lbs_25VLS := 0;
  BaseCostSteps.prerule_ploading_lbs_35VLS := 0;
  BaseCostSteps.postrule_ploading_lbs_5VLS := 0;
  BaseCostSteps.postrule_ploading_lbs_15VLS := 0;
  BaseCostSteps.postrule_ploading_lbs_25VLS := 0;
  BaseCostSteps.postrule_ploading_lbs_35VLS := 0;
  BaseCostSteps.incr_ploading_lbs_5VLS := 0;
  BaseCostSteps.incr_ploading_lbs_15VLS := 0;
  BaseCostSteps.incr_ploading_lbs_25VLS := 0;
  BaseCostSteps.incr_ploading_lbs_35VLS := 0;
  BaseCostSteps.count_incr_ploading_lbs_5VLS := 0;
  BaseCostSteps.count_incr_ploading_lbs_15VLS := 0;
  BaseCostSteps.count_incr_ploading_lbs_25VLS := 0;
  BaseCostSteps.count_incr_ploading_lbs_35VLS := 0;
end
else
begin
  ScenCostSteps.HasLSLRCostVLS := false;
  ScenCostSteps.HasCCTCostVLS := false;
  ScenCostSteps.LSLReplacedVLS := 0;
  ScenCostSteps.LSLReplacedPopVLS := 0;
  ScenCostSteps.LSLReplacedMandatoryVLS := 0;
  ScenCostSteps.LSLReplacedVoluntaryVLS := 0;
  ScenCostSteps.LSLReplacedPopMandatoryVLS := 0;
```

```
  ScenCostSteps.LSLReplacedPopVoluntaryVLS := 0;
  ScenCostSteps.LSLRequestedVLS := 0;
  ScenCostSteps.LSLRequestedPopVLS := 0;
  ScenCostSteps.CCTInstalledVLS := false;
  ScenCostSteps.CCTAdjustedVLS := false;
  ScenCostSteps.CCTAdjustedVLS_ale := false;
  ScenCostSteps.CCTAdjustedVLS_tle := false;
  ScenCostSteps.CCTInstalledPopVLS := 0;
  ScenCostSteps.CCTAdjustedPopVLS := 0;
  ScenCostSteps.CCTAdjustedPopVLS_ale := 0;
  ScenCostSteps.CCTAdjustedPopVLS_tle := 0;
  ScenCostSteps.SystemAFlowVLS := AFlow;
  ScenCostSteps.CCTExistingVLS := false;
  ScenCostSteps.HasFindAndFixCostVLS := false;
  ScenCostSteps.POUInstalledVLS := false;
  ScenCostSteps.CCTExistingPopVLS := 0;
  ScenCostSteps.HasFindAndFixCostPopVLS := 0;
  ScenCostSteps.POUInstalledPopVLS := 0;

  ScenCostSteps.POTWCostVLS := 0;

  ScenCostSteps.prerule_ploading_lbs_5VLS := 0;
  ScenCostSteps.prerule_ploading_lbs_15VLS := 0;
  ScenCostSteps.prerule_ploading_lbs_25VLS := 0;
  ScenCostSteps.prerule_ploading_lbs_35VLS := 0;
  ScenCostSteps.postrule_ploading_lbs_5VLS := 0;
  ScenCostSteps.postrule_ploading_lbs_15VLS := 0;
  ScenCostSteps.postrule_ploading_lbs_25VLS := 0;
  ScenCostSteps.postrule_ploading_lbs_35VLS := 0;
  ScenCostSteps.incr_ploading_lbs_5VLS := 0;
  ScenCostSteps.incr_ploading_lbs_15VLS := 0;
  ScenCostSteps.incr_ploading_lbs_25VLS := 0;
  ScenCostSteps.incr_ploading_lbs_35VLS := 0;
  ScenCostSteps.count_incr_ploading_lbs_5VLS := 0;
  ScenCostSteps.count_incr_ploading_lbs_15VLS := 0;
  ScenCostSteps.count_incr_ploading_lbs_25VLS := 0;
  ScenCostSteps.count_incr_ploading_lbs_35VLS := 0;
end;

Filename := DataPath + 'VLSEntryPointValues.xlsx';

iNumEpEntries := NumEpEntries;
CostingData.EntryPoints := NumEpEntries;

Xls := TXlsFile.Create(Filename, False);
Xls.ActiveSheetByName := 'Inputs';

{
```

```
  VLSEntryPointValues.xlsx
  1 PWSID
  2 p_b3
  3 p_b3_r
  4 Baselineph_wocct
  5 Baselineph_wph
  6 Baselineph_wpo4ph
  7 Baselineph_woph
  8 BaselinePO4Dose
  9 CCT
 10 LSL
 11 NumberEPs
 12 pbasephpo4
 13 pbaseph
 14 pbasepo4
 15 Population
 //16 Connections
 //17 NumberLSLs
}

for r := 2 to Xls.RowCount do
begin
  if Xls.GetStringFromCell(r,1) = Copy(CostingData.PWSId,1,9) then
  begin
    inc(idcnt);
    VLSEpWorkbook.p_b3 := -1;
    VLSEPWorkbook.baselinePH_woCCT := -1;
    VLSEpWorkbook.baselinePH_wPh := -1;
    VLSEpWorkbook.baselinePH_woPh := -1;
    VLSEpWorkbook.baselinePH_wPO4Ph := -1;
    VLSEpWorkbook.baselinePO4Dose := -1;
    VLSEpWorkbook.CCT := -1;
    VLSEpWorkbook.NumberLSLs := -1;
    VLSEpWorkbook.LSL := -1;

    if VarIsNumeric(xls.GetCellValue(r, 2).AsVariant) then
      VLSEpWorkbook.p_b3 := xls.GetCellValue(r, 2).AsVariant;

    if VarIsNumeric(xls.GetCellValue(r, 4).AsVariant) then
      VLSEpWorkbook.baselinePH_woCCT := xls.GetCellValue(r, 4).AsVariant
    else
      VLSEpWorkbook.baselinePH_woCCT := CostingData.BaselinePH_woCCT;

    if VarIsNumeric(xls.GetCellValue(r, 5).AsVariant) then
      VLSEpWorkbook.baselinePH_wPh := xls.GetCellValue(r, 5).AsVariant
    else
      VLSEpWorkbook.baselinePH_wPh := CostingData.BaselinePH_wph;
```

```
      if VarIsNumeric(xls.GetCellValue(r, 6).AsVariant) then
        VLSEpWorkbook.baselinePH_wPO4Ph := xls.GetCellValue(r, 6).AsVariant
      else
        VLSEpWorkbook.baselinePH_wPO4Ph := CostingData.BaselinePH_wPO4ph;

      if VarIsNumeric(xls.GetCellValue(r, 7).AsVariant) then
        VLSEpWorkbook.baselinePH_woPh := xls.GetCellValue(r, 7).AsVariant
      else
        VLSEpWorkbook.baselinePH_woPh := CostingData.BaselinePH_woph;

      if VarIsNumeric(xls.GetCellValue(r, 8).AsVariant) then
        VLSEpWorkbook.baselinePO4Dose := xls.GetCellValue(r, 8).AsVariant
      else
        VLSEpWorkbook.baselinePO4Dose := CostingData.BaselinePO4Dose;

      if VarIsNumeric(xls.GetCellValue(r, 9).AsVariant) then
        VLSEpWorkbook.CCT := xls.GetCellValue(r, 9).AsVariant
      else
        VLSEpWorkbook.CCT := CostingData.CCT;

      if VarIsNumeric(xls.GetCellValue(r, 10).AsVariant) then
        VLSEpWorkbook.LSL := xls.GetCellValue(r, 10).AsVariant;

      VLSEpWorkbook.NumberEPs := xls.GetCellValue(r, 11).AsVariant;

      if VarIsNumeric(xls.GetCellValue(r, 12).AsVariant) then
        VLSEpWorkbook.pbasephpo4 := xls.GetCellValue(r, 12).AsVariant
      else
        VLSEpWorkbook.pbasephpo4 := CostingData.CCTBoth ;

      if VarIsNumeric(xls.GetCellValue(r, 13).AsVariant) then
        VLSEpWorkbook.pbaseph := xls.GetCellValue(r, 13).AsVariant
      else
        VLSEpWorkbook.pbaseph := CostingData.CCTPH ;

      if VarIsNumeric(xls.GetCellValue(r, 14).AsVariant) then
        VLSEpWorkbook.pbasepo4 := xls.GetCellValue(r, 14).AsVariant
      else
        VLSEpWorkbook.pbasepo4 := CostingData.CCTPO4 ;

      VLSEpWorkbook.Population := xls.GetCellValue(r, 15).AsVariant;

      VLSEpWorkbook.Connections := Round((CostingData.Connections /
CostingData.Population) * VLSEpWorkbook.Population);

      if iNumLSLEps > 0 then
        VLSEpWorkbook.NumberLSLs := Round(CostingData.NumberLSLs / iNumLSLEps)
      else
```

```
        VLSEpWorkbook.NumberLSLs := Round(CostingData.NumberLSLs);

      Config.GetFlowsEp(VLSEpWorkbook.NumberEPs, CostingData.Ownership,
CostingData.SourceWater,
                        VLSEpWorkbook.Population,
                        VLSEpWorkbook.AFlowEp, VLSEpWorkbook.DFlowEp);

      CCTCostEquations.DFlowEP := VLSEpWorkbook.DFlowEp;
      CCTCostEquations.AFlowEP := VLSEpWorkbook.AFlowEp;
      CCTCostEquations.EntryPoints := VLSEpWorkbook.NumberEPs;
      CCTCostEquations.iSystemSize := CostingData.SystemSize;
      CCTCostEquations.iSourceWater := CostingData.SourceWater;

      CCTCostEquations.pbaseph := VLSEpWorkbook.pbaseph;
      CCTCostEquations.pbasepo4 := VLSEpWorkbook.pbasepo4;
      CCTCostEquations.pbasephpo4 := VLSEpWorkbook.pbasephpo4;

      CCTCostEquations.iBaselinepo4dose := VLSEpWorkbook.BaselinePO4Dose;
      CCTCostEquations.iBaselineph_wph := VLSEpWorkbook.BaselinePH_wph;
      CCTCostEquations.iBaselineph_woph := VLSEpWorkbook.BaselinePH_woph;
      CCTCostEquations.iBaselineph_wocct := VLSEpWorkbook.baselinePH_woCCT;
      CCTCostEquations.iBaselineph_wpo4ph := VLSEpWorkbook.Baselineph_wpo4ph;

      CostingData.SamplingWeight := CostingData.SamplingWeight / iNumEpEntries;

      if option = 'Baseline' then
      begin
        BaseCostSteps.LSLReplaced := 0;
        BaseCostSteps.LSLRequested := 0;
        BaseCostSteps.LSLReplacedMandatory := 0;
        BaseCostSteps.LSLReplacedVoluntary := 0;
        BaseCostSteps.SetVariablesAndCalculateVLS(CostingData, BAddCostingData,
VLSEpWorkbook, True, option,
                                                  CCTCostEquations,
                                                  CostingData.fBaseVars,
SchoolSampData, idcnt=1);
        if BaseCostSteps.LSLReplaced > 0 then
        begin
          BaseCostSteps.LSLReplacedVLS  := BaseCostSteps.LSLReplacedVLS +
BaseCostSteps.LSLReplaced;
          BaseCostSteps.LSLReplacedPopVLS := BaseCostSteps.LSLReplacedPopVLS +
                                             round(BaseCostSteps.LSLReplaced *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplacedMandatoryVLS  :=
BaseCostSteps.LSLReplacedMandatoryVLS + BaseCostSteps.LSLReplacedMandatory;
          BaseCostSteps.LSLReplacedVoluntaryVLS  :=
BaseCostSteps.LSLReplacedVoluntaryVLS + BaseCostSteps.LSLReplacedVoluntary;
```

```
          BaseCostSteps.LSLReplacedPopMandatoryVLS :=
BaseCostSteps.LSLReplacedPopMandatoryVLS +
               round(BaseCostSteps.LSLReplacedMandatory *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
          BaseCostSteps.LSLReplacedPopVoluntaryVLS :=
BaseCostSteps.LSLReplacedPopVoluntaryVLS +
               round(BaseCostSteps.LSLReplacedVoluntary *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
        end;

        if BaseCostSteps.LSLRequested > 0 then
        begin
          BaseCostSteps.LSLRequestedVLS  := BaseCostSteps.LSLRequestedVLS +
BaseCostSteps.LSLRequested;
          BaseCostSteps.LSLRequestedPopVLS := BaseCostSteps.LSLRequestedPopVLS +
                                                 round(BaseCostSteps.LSLRequested *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
        end;

        if BaseCostSteps.CCTInstalled then
        begin
          BaseCostSteps.CCTInstalledVLS := true;
          BaseCostSteps.CCTInstalledPopVLS := BaseCostSteps.CCTInstalledPopVLS +
VLSEpWorkbook.Population;
        end;

        if BaseCostSteps.CCTAdjusted then
        begin
          BaseCostSteps.CCTAdjusted := true;
          BaseCostSteps.CCTAdjustedPopVLS := BaseCostSteps.CCTAdjustedPopVLS +
VLSEpWorkbook.Population;
          BaseCostSteps.CCTAdjustedPopVLS_ale := BaseCostSteps.CCTAdjustedPopVLS;
        end;
        BaseCostSteps.CCTAdjustedPopVLS_tle := 0;

        if BaseCostSteps.CCTExisting then
        begin
          BaseCostSteps.CCTExistingVLS := true;
          BaseCostSteps.CCTExistingPopVLS := BaseCostSteps.CCTExistingPopVLS +
VLSEpWorkbook.Population;
        end;
        if BaseCostSteps.HasFindAndFixCostVLS then
        begin
          BaseCostSteps.HasFindAndFixCost := true;
          BaseCostSteps.HasFindAndFixCostPopVLS :=
BaseCostSteps.HasFindAndFixCostPopVLS + VLSEpWorkbook.Population;
        end;
```

```
        if BaseCostSteps.POUInstalled then
        begin
          BaseCostSteps.POUInstalledVLS := true;
          BaseCostSteps.POUInstalledPopVLS := BaseCostSteps.POUInstalledPopVLS +
VLSEpWorkbook.Population;
        end;

        BaseCostSteps.POTWCostVLS := BaseCostSteps.POTWCostVLS +
BaseCostSteps.POTWCost;

        BaseCostSteps.prerule_ploading_lbs_5VLS :=
BaseCostSteps.prerule_ploading_lbs_5VLS + BaseCostSteps.prerule_ploading_lbs_5;
        BaseCostSteps.prerule_ploading_lbs_15VLS :=
BaseCostSteps.prerule_ploading_lbs_15VLS + BaseCostSteps.prerule_ploading_lbs_15;
        BaseCostSteps.prerule_ploading_lbs_25VLS :=
BaseCostSteps.prerule_ploading_lbs_25VLS + BaseCostSteps.prerule_ploading_lbs_25;
        BaseCostSteps.prerule_ploading_lbs_35VLS :=
BaseCostSteps.prerule_ploading_lbs_35VLS + BaseCostSteps.prerule_ploading_lbs_35;
        BaseCostSteps.postrule_ploading_lbs_5VLS :=
BaseCostSteps.postrule_ploading_lbs_5VLS + BaseCostSteps.postrule_ploading_lbs_5;
        BaseCostSteps.postrule_ploading_lbs_15VLS :=
BaseCostSteps.postrule_ploading_lbs_15VLS + BaseCostSteps.postrule_ploading_lbs_15;
        BaseCostSteps.postrule_ploading_lbs_25VLS :=
BaseCostSteps.postrule_ploading_lbs_25VLS + BaseCostSteps.postrule_ploading_lbs_25;
        BaseCostSteps.postrule_ploading_lbs_35VLS :=
BaseCostSteps.postrule_ploading_lbs_35VLS + BaseCostSteps.postrule_ploading_lbs_35;
        BaseCostSteps.incr_ploading_lbs_5VLS := BaseCostSteps.incr_ploading_lbs_5VLS
+ BaseCostSteps.incr_ploading_lbs_5;
        BaseCostSteps.incr_ploading_lbs_15VLS :=
BaseCostSteps.incr_ploading_lbs_15VLS + BaseCostSteps.incr_ploading_lbs_15;
        BaseCostSteps.incr_ploading_lbs_25VLS :=
BaseCostSteps.incr_ploading_lbs_25VLS + BaseCostSteps.incr_ploading_lbs_25;
        BaseCostSteps.incr_ploading_lbs_35VLS :=
BaseCostSteps.incr_ploading_lbs_35VLS + BaseCostSteps.incr_ploading_lbs_35;
        if BaseCostSteps.count_incr_ploading_lbs_5 = 1 then
BaseCostSteps.count_incr_ploading_lbs_5VLS := 1;
        if BaseCostSteps.count_incr_ploading_lbs_15 = 1 then
BaseCostSteps.count_incr_ploading_lbs_15VLS := 1;
        if BaseCostSteps.count_incr_ploading_lbs_25 = 1 then
BaseCostSteps.count_incr_ploading_lbs_25VLS := 1;
        if BaseCostSteps.count_incr_ploading_lbs_35 = 1 then
BaseCostSteps.count_incr_ploading_lbs_35VLS := 1;
      end
      else
      begin
        ScenCostSteps.LSLReplaced := 0;
        ScenCostSteps.LSLRequested := 0;
        ScenCostSteps.LSLReplacedMandatory := 0;
```

```
        ScenCostSteps.LSLReplacedVoluntary := 0;
        ScenCostSteps.SetVariablesAndCalculateVLS(CostingData,
SAddCostingData,VLSEpWorkbook, True, option,
                                                  CCTCostEquations,
                                                  CostingData.fScenVars,
SchoolSampData, idcnt=1);
        if ScenCostSteps.LSLReplaced > 0 then
        begin
          ScenCostSteps.LSLReplacedVLS  := ScenCostSteps.LSLReplacedVLS +
ScenCostSteps.LSLReplaced;
          ScenCostSteps.LSLReplacedPopVLS := ScenCostSteps.LSLReplacedPopVLS +
                                             round(ScenCostSteps.LSLReplaced *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplacedMandatoryVLS  :=
ScenCostSteps.LSLReplacedMandatoryVLS + ScenCostSteps.LSLReplacedMandatory;
          ScenCostSteps.LSLReplacedVoluntaryVLS  :=
ScenCostSteps.LSLReplacedVoluntaryVLS + ScenCostSteps.LSLReplacedVoluntary;

          ScenCostSteps.LSLReplacedPopMandatoryVLS :=
ScenCostSteps.LSLReplacedPopMandatoryVLS +
               round(ScenCostSteps.LSLReplacedMandatory *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
          ScenCostSteps.LSLReplacedPopVoluntaryVLS :=
ScenCostSteps.LSLReplacedPopVoluntaryVLS +
               round(ScenCostSteps.LSLReplacedVoluntary *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
        end;

        if ScenCostSteps.LSLRequested > 0 then
        begin
          ScenCostSteps.LSLRequestedVLS  := ScenCostSteps.LSLRequestedVLS +
ScenCostSteps.LSLRequested;
          ScenCostSteps.LSLRequestedPopVLS := ScenCostSteps.LSLRequestedPopVLS +
                                              round(ScenCostSteps.LSLRequested *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
        end;

        if ScenCostSteps.CCTInstalled then
        begin
          ScenCostSteps.CCTInstalledVLS := true;
          ScenCostSteps.CCTInstalledPopVLS := ScenCostSteps.CCTInstalledPopVLS +
VLSEpWorkbook.Population;
        end;

        if ScenCostSteps.CCTAdjusted then
        begin
          ScenCostSteps.CCTAdjustedVLS := true;
```

```
          ScenCostSteps.CCTAdjustedPopVLS := ScenCostSteps.CCTAdjustedPopVLS +
VLSEpWorkbook.Population;
        end;
        if ScenCostSteps.CCTAdjusted_ale then
        begin
          ScenCostSteps.CCTAdjustedVLS_ale := true;
          ScenCostSteps.CCTAdjustedPopVLS_ale := ScenCostSteps.CCTAdjustedPopVLS_ale
+ VLSEpWorkbook.Population;
        end;
        if ScenCostSteps.CCTAdjusted_tle then
        begin
          ScenCostSteps.CCTAdjustedVLS_tle := true;
          ScenCostSteps.CCTAdjustedPopVLS_tle := ScenCostSteps.CCTAdjustedPopVLS_tle
+ VLSEpWorkbook.Population;
        end;

        if ScenCostSteps.CCTExisting then
        begin
          ScenCostSteps.CCTExistingVLS := true;
          ScenCostSteps.CCTExistingPopVLS := ScenCostSteps.CCTExistingPopVLS +
VLSEpWorkbook.Population;
        end;
        if ScenCostSteps.HasFindAndFixCost then
        begin
          ScenCostSteps.HasFindAndFixCostVLS := true;
          ScenCostSteps.HasFindAndFixCostPopVLS :=
ScenCostSteps.HasFindAndFixCostPopVLS + VLSEpWorkbook.Population;
        end;
        if ScenCostSteps.POUInstalled then
        begin
          ScenCostSteps.POUInstalledVLS := true;
          ScenCostSteps.POUInstalledPopVLS := ScenCostSteps.POUInstalledPopVLS +
VLSEpWorkbook.Population;
        end;

        ScenCostSteps.POTWCostVLS := ScenCostSteps.POTWCostVLS +
ScenCostSteps.POTWCost;

        ScenCostSteps.prerule_ploading_lbs_5VLS :=
ScenCostSteps.prerule_ploading_lbs_5VLS + ScenCostSteps.prerule_ploading_lbs_5;
        ScenCostSteps.prerule_ploading_lbs_15VLS :=
ScenCostSteps.prerule_ploading_lbs_15VLS + ScenCostSteps.prerule_ploading_lbs_15;
        ScenCostSteps.prerule_ploading_lbs_25VLS :=
ScenCostSteps.prerule_ploading_lbs_25VLS + ScenCostSteps.prerule_ploading_lbs_25;
        ScenCostSteps.prerule_ploading_lbs_35VLS :=
ScenCostSteps.prerule_ploading_lbs_35VLS + ScenCostSteps.prerule_ploading_lbs_35;
        ScenCostSteps.postrule_ploading_lbs_5VLS :=
ScenCostSteps.postrule_ploading_lbs_5VLS + ScenCostSteps.postrule_ploading_lbs_5;
```

```
        ScenCostSteps.postrule_ploading_lbs_15VLS :=
ScenCostSteps.postrule_ploading_lbs_15VLS + ScenCostSteps.postrule_ploading_lbs_15;
        ScenCostSteps.postrule_ploading_lbs_25VLS :=
ScenCostSteps.postrule_ploading_lbs_25VLS + ScenCostSteps.postrule_ploading_lbs_25;
        ScenCostSteps.postrule_ploading_lbs_35VLS :=
ScenCostSteps.postrule_ploading_lbs_35VLS + ScenCostSteps.postrule_ploading_lbs_35;
        ScenCostSteps.incr_ploading_lbs_5VLS := ScenCostSteps.incr_ploading_lbs_5VLS
+ ScenCostSteps.incr_ploading_lbs_5;
        ScenCostSteps.incr_ploading_lbs_15VLS :=
ScenCostSteps.incr_ploading_lbs_15VLS + ScenCostSteps.incr_ploading_lbs_15;
        ScenCostSteps.incr_ploading_lbs_25VLS :=
ScenCostSteps.incr_ploading_lbs_25VLS + ScenCostSteps.incr_ploading_lbs_25;
        ScenCostSteps.incr_ploading_lbs_35VLS :=
ScenCostSteps.incr_ploading_lbs_35VLS + ScenCostSteps.incr_ploading_lbs_35;
        if ScenCostSteps.count_incr_ploading_lbs_5 = 1 then
ScenCostSteps.count_incr_ploading_lbs_5VLS := 1;
        if ScenCostSteps.count_incr_ploading_lbs_15 = 1 then
ScenCostSteps.count_incr_ploading_lbs_15VLS := 1;
        if ScenCostSteps.count_incr_ploading_lbs_25 = 1 then
ScenCostSteps.count_incr_ploading_lbs_25VLS := 1;
        if ScenCostSteps.count_incr_ploading_lbs_35 = 1 then
ScenCostSteps.count_incr_ploading_lbs_35VLS := 1;
      end;
    end;
  end;

  if option = 'Baseline' then
    BaseCostSteps.Annualize(CostingData.CostCapital)
  else
    ScenCostSteps.Annualize(CostingData.CostCapital);

  FreeAndNil(Xls);
end;

end.
```