```
unit LCRCosts;

interface

uses Classes, SysUtils, StrUtils, LCRGlobals, LCRConfig,
     LCRMetricCollector, SafewaterUncertBucket, CostingSteps,
     LCRAggregationConsts,LCRCostVars,CodeSiteLogging,
     DB, ADODB, Generics.Collections, CCTCostEquations, StateSchoolSampData
     {$ifndef CONSOLE}
     ,dialogs
     {$endif}


     ;
type

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

  TLCRCosts = class(TObject)
    private

      //TODO these should created outside and shared -
      LCRBaseCostVars, 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, LSLReplacementPopulation: double;
      LSLReplacementMandatory, LSLReplacementVoluntary: double;
      LSLReplacementPopMandatory, LSLReplacementPopVoluntary: double;
      LSLReplacementRequested, LSLReplacementPopRequested: double;
      AddModifyCCT: double;
      AddModifyCCTPopulation: double;

      // Total
      LSLReplaced, LSLReplacedMandatory, LSLReplacedVoluntary: double;
      LSLReplacedRequested: double;
      DeferralAny, DeferralPct, DeferralCap: double;

      LSLReplaceFullMandatory, LSLReplaceFullPopMandatory: double;
      LSLReplacePartialMandatory, LSLReplacePartialPopMandatory: double;
      LSLGalPrevMandatory, LSLGalPrevPopMandatory: double;
      LSLLeadConMandatory, LSLLeadConPopMandatory: double;
      LSLGalPrevLeadConMandatory, LSLGalPrevLeadConPopMandatory: double;

      LSLReplaceFullVoluntary, LSLReplaceFullPopVoluntary: double;
      LSLReplacePartialVoluntary, LSLReplacePartialPopVoluntary: double;
      LSLGalPrevVoluntary, LSLGalPrevPopVoluntary: double;
      LSLLeadConVoluntary, LSLLeadConPopVoluntary: double;
      LSLGalPrevLeadConVoluntary, LSLGalPrevLeadConPopVoluntary: double;

      // Non-State
      LSLReplacedNS, LSLReplacedMandatoryNS, LSLReplacedVoluntaryNS: double;

      LSLReplaceFullMandatoryNS, LSLReplaceFullPopMandatoryNS: double;
      LSLReplacePartialMandatoryNS, LSLReplacePartialPopMandatoryNS: double;
      LSLGalPrevMandatoryNS, LSLGalPrevPopMandatoryNS: double;
      LSLLeadConMandatoryNS, LSLLeadConPopMandatoryNS: double;
      LSLGalPrevLeadConMandatoryNS, LSLGalPrevLeadConPopMandatoryNS: double;
```

```
      LSLReplaceFullVoluntaryNS, LSLReplaceFullPopVoluntaryNS: double;
      LSLReplacePartialVoluntaryNS, LSLReplacePartialPopVoluntaryNS: double;
      LSLGalPrevVoluntaryNS, LSLGalPrevPopVoluntaryNS: double;
      LSLLeadConVoluntaryNS, LSLLeadConPopVoluntaryNS: double;
      LSLGalPrevLeadConVoluntaryNS, LSLGalPrevLeadConPopVoluntaryNS: 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;

      _TotalCostCapBase,_TotalCostCapOption : double; //Used for proxies..

      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;
      ToBin1Count: double;
      LSLReplacementSystemAncillary, LSLReplacementSystemCapital,
SystemLeadModifyCCT,
           SystemLeadModifyCCTAncillary, SystemSchoolDayCareTapSampling,
LSLReplacementProgramAncillary: double;
      LSLReplacementSystemAncillaryPWS, LSLReplacementSystemCapitalPWS,
```

```
SystemLeadModifyCCTPWS,
           SystemLeadModifyCCTAncillaryPWS, SystemSchoolDayCareTapSamplingPWS,
LSLReplacementProgramAncillaryPWS: 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;

      AdjPWSLSLRepPcts: array[1..5] of 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;

      PubNewPathO: double;

      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: TStateSchoolSampData; pws_state:
string);
      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 := GetVE(acTotalEdu);
  TotalEDUPWS := GetVE(acTotalEduPWS);

  LSLReplacementSystemAncillary := GetVE(acLSLReplacementSystemAncillary);
  LSLReplacementSystemAncillaryPWS := GetVE(acLSLReplacementSystemAncillaryPWS);
  LSLReplacementSystemCapital := GetVE(acLSLReplacementSystemCapital);
  LSLReplacementSystemCapitalPWS := GetVE(acLSLReplacementSystemCapitalPWS);
  SystemLeadModifyCCT := GetVE(acSystemLeadModifyCCT);
  SystemLeadModifyCCTPWS := GetVE(acSystemLeadModifyCCTPWS);
  SystemLeadModifyCCTAncillary := GetVE(acSystemLeadModifyCCTAncillary);
  SystemLeadModifyCCTAncillaryPWS := GetVE(acSystemLeadModifyCCTAncillaryPWS);
  SystemSchoolDayCareTapSampling := GetVE(acSystemSchoolDayCareTapSampling);
  SystemSchoolDayCareTapSamplingPWS := GetVE(acSystemSchoolDayCareTapSamplingPWS);
  LSLReplacementProgramAncillary := GetVE(acLSLReplacementProgramAncillary) +
GetVE(acLSLReplacementSystemAncillary);
  LSLReplacementProgramAncillaryPWS := GetVE(acLSLReplacementProgramAncillaryPWS) +
GetVE(acLSLReplacementSystemAncillaryPWS);

  TotalCostCap := GetVE(acTotalCostCap);
  _TotalCostCapBase := GetVESingle(acTotalCostCap,true);
  _TotalCostCapOption := GetVESingle(acTotalCostCap,false);
```

```
  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);

  // LCR Baseline database
  if (aConfig.UseLCRBaseline = 1) then
  begin
    ADOCon:=TADOCOnnection.Create(nil);
    ADOCon.ConnectionString:=format(ADOConStr,[aConfig.LCRBaselineDB]);
    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;

LCRBaseCostVars:=TCostVars.Create(qDesc,qData,ChangeFileExt(aConfig.LCRBaselineDB,'.
xlsm'), rrAlt);
    qData.Close;
    qDesc.Close;
    AdoCon.Close;
    qData.Free;
    qDesc.Free;
    ADOCon.Free;
  end;

  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;
  if aConfig.UseLCRBaseline = 0 then

BaseCostVars:=TCostVars.Create(qDesc,qData,ChangeFileExt(aConfig.BaseVarData,'.xlsm'
), rrBase)
  else

BaseCostVars:=TCostVars.Create(qDesc,qData,ChangeFileExt(aConfig.BaseVarData,'.xlsm'
), rrBase, LCRBaseCostVars);
  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'
), rrScen, BaseCostVars);

  AdoCon.Close;
  qData.Close;
  qDesc.Close;
  qData.Free;
  qDesc.Free;
  ADOCon.Free;

  AggregatedResults := TDictionary<string, TAggInputRec>.Create;

  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.SetDefault();
    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.SetDefault();
    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
      AggInput.SetDefault();


    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
      AggInput.SetDefault();

    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
      AggInput.SetDefault();

    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(@ToBin1Count,@DummyProb,nil,'To Bin 1
Count',mtInfo,False,False,False,Config.OptionName,0,True);

  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 Replaced Any
Type Population',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementPopMandatory,@DummyProb,nil,'LSL Replaced
Any Type Population Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementPopVoluntary,@DummyProb,nil,'LSL Replaced
Any Type 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(@LSLReplacementSystemAncillary,@DummyProb,nil,'LSL
Replacement System Ancillary
Total',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementSystemCapital,@DummyProb,nil,'LSL
Replacement System Capital
Total',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@SystemLeadModifyCCT,@DummyProb,nil,'System Lead Modify
CCT Total',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@SystemLeadModifyCCTAncillary,@DummyProb,nil,'System Lead
Modify CCT Ancillary Total',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@SystemSchoolDayCareTapSampling,@DummyProb,nil,'System
School and Day Care Tap
Sampling',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementSystemAncillaryPWS,@DummyProb,nil,'LSL
Replacement System Ancillary Total
PWS',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementSystemCapitalPWS,@DummyProb,nil,'LSL
Replacement System Capital Total
PWS',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@SystemLeadModifyCCTPWS,@DummyProb,nil,'System Lead Modify
CCT Total PWS',mtInfo,False,False,False,Config.OptionName,0,True);
```

```
  Outputs.AddOutputMetric(@SystemLeadModifyCCTAncillaryPWS,@DummyProb,nil,'System
Lead Modify CCT Ancillary Total
PWS',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@SystemSchoolDayCareTapSamplingPWS,@DummyProb,nil,'System
School and Day Care Tap Sampling
PWS',mtInfo,False,False,False,Config.OptionName,0,True);

  Outputs.AddOutputMetric(@LSLReplacementProgramAncillary,@DummyProb,nil,'LSL
Replacement Program Ancillary
Total',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacementProgramAncillaryPWS,@DummyProb,nil,'LSL
Replacement Program Ancillary Total
PWS',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(@DeferralAny,@DummyProb,nil,'Number of PWS with LSLR
Deferral Any',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@DeferralPct,@DummyProb,nil,'Number of PWS with LSLR
Deferral Percent',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@DeferralCap,@DummyProb,nil,'Number of PWS with LSLR
Deferral Cap',mtInfo,False,False,False,Config.OptionName,0,True);

  Outputs.AddOutputMetric(@PubNewPathO,@DummyProb,nil,'Number of PWS with new
path',mtInfo,False,False,False,Config.OptionName,0,True);

  // Total LSL
  Outputs.AddOutputMetric(@LSLReplaced,@DummyProb,nil,'Total LSL
Replaced',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplaceFullMandatory,@DummyProb,nil,'Total LSL
Replaced Full Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplaceFullPopMandatory,@DummyProb,nil,'Total LSL
Replaced Full Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialMandatory,@DummyProb,nil,'Total LSL
Replaced Partial Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialPopMandatory,@DummyProb,nil,'Total LSL
Replaced Partial Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevMandatory,@DummyProb,nil,'Total LSL Replaced
Gal Prev Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevPopMandatory,@DummyProb,nil,'Total LSL Replaced
Gal Prev Population Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConMandatory,@DummyProb,nil,'Total LSL Replaced
LeadCon Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConPopMandatory,@DummyProb,nil,'Total LSL Replaced
LeadCon Population Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevLeadConMandatory,@DummyProb,nil,'Total LSL
Replaced GalPrev LeadCon
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
```

```
  Outputs.AddOutputMetric(@LSLGalPrevLeadConPopMandatory,@DummyProb,nil,'Total LSL
Replaced GalPrev LeadCon Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);

  // Non-State LSL
  Outputs.AddOutputMetric(@LSLReplacedNS,@DummyProb,nil,'Non-State LSL
Replaced',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplaceFullMandatoryNS,@DummyProb,nil,'Non-State LSL
Replaced Full Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplaceFullPopMandatoryNS,@DummyProb,nil,'Non-State
LSL Replaced Full Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialMandatoryNS,@DummyProb,nil,'Non-State
LSL Replaced Partial Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialPopMandatoryNS,@DummyProb,nil,'Non-State
LSL Replaced Partial Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevMandatoryNS,@DummyProb,nil,'Non-State LSL
Replaced Gal Prev Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevPopMandatoryNS,@DummyProb,nil,'Non-State LSL
Replaced Gal Prev Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConMandatoryNS,@DummyProb,nil,'Non-State LSL
Replaced LeadCon Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConPopMandatoryNS,@DummyProb,nil,'Non-State LSL
Replaced LeadCon Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevLeadConMandatoryNS,@DummyProb,nil,'Non-State
LSL Replaced GalPrev LeadCon
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevLeadConPopMandatoryNS,@DummyProb,nil,'Non-State
LSL Replaced GalPrev LeadCon Population
Mandatory',mtInfo,False,False,False,Config.OptionName,0,True);

  // Total LSL
  Outputs.AddOutputMetric(@LSLReplaceFullVoluntary,@DummyProb,nil,'Total LSL
Replaced Full Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplaceFullPopVoluntary,@DummyProb,nil,'Total LSL
Replaced Full Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialVoluntary,@DummyProb,nil,'Total LSL
Replaced Partial Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialPopVoluntary,@DummyProb,nil,'Total LSL
Replaced Partial Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevVoluntary,@DummyProb,nil,'Total LSL Replaced
Gal Prev Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevPopVoluntary,@DummyProb,nil,'Total LSL Replaced
Gal Prev Population Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConVoluntary,@DummyProb,nil,'Total LSL Replaced
```

```
LeadCon Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConPopVoluntary,@DummyProb,nil,'Total LSL Replaced
LeadCon Population Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevLeadConVoluntary,@DummyProb,nil,'Total LSL
Replaced GalPrev LeadCon
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevLeadConPopVoluntary,@DummyProb,nil,'Total LSL
Replaced GalPrev LeadCon Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);

  // Non-State LSL
  Outputs.AddOutputMetric(@LSLReplaceFullVoluntaryNS,@DummyProb,nil,'Non-State LSL
Replaced Full Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplaceFullPopVoluntaryNS,@DummyProb,nil,'Non-State
LSL Replaced Full Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialVoluntaryNS,@DummyProb,nil,'Non-State
LSL Replaced Partial Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLReplacePartialPopVoluntaryNS,@DummyProb,nil,'Non-State
LSL Replaced Partial Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevVoluntaryNS,@DummyProb,nil,'Non-State LSL
Replaced Gal Prev Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevPopVoluntaryNS,@DummyProb,nil,'Non-State LSL
Replaced Gal Prev Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConVoluntaryNS,@DummyProb,nil,'Non-State LSL
Replaced LeadCon Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLLeadConPopVoluntaryNS,@DummyProb,nil,'Non-State LSL
Replaced LeadCon Population
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevLeadConVoluntaryNS,@DummyProb,nil,'Non-State
LSL Replaced GalPrev LeadCon
Voluntary',mtInfo,False,False,False,Config.OptionName,0,True);
  Outputs.AddOutputMetric(@LSLGalPrevLeadConPopVoluntaryNS,@DummyProb,nil,'Non-State
LSL Replaced GalPrev LeadCon Population
Voluntary',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.OutputBenefitMoveBins(Config.RunName);
  ScenCostSteps.OutputBenefitMoveBins(Config.RunName);
  BaseCostSteps.Free;
  ScenCostSteps.Free;
  BaseCostVars.Free;
  ScenCostVars.Free;
  if Config.UseLCRBaseline = 1 then LCRBaseCostVars.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: TStateSchoolSampData; pws_state:
string);
var
  i : integer;
  dPWSid: string;
  DictItem: TPair<string, TAggInputRec>;
  HHConsumption: double;
  BaseHasIt, ScenHasIt: integer;
  prtDebug, VLSystem: boolean;

  total_replacements, total_replacements_Base, total_replacements_Opt: double;
  total_replacements_mand, total_replacements_mand_Base,
total_replacements_mand_Opt: double;
  total_replacements_vol, total_replacements_vol_Base, total_replacements_vol_Opt:
double;
  SchoolSampDataRec: TSchoolSampDataRec;
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 > 0 then Inc(HasLSLCnt)
  else Inc(NoLSLCnt);
```

```
  if UserSeeds then RandSeed:=RndSeed;

  SchoolSampDataRec := SchoolSampData.GetStateSchoolSampData(pws_state);

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

    BaseCostSteps.SetVariablesAndCalculate(CostingData, 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,
SchoolSampDataRec,
                                           pws_state);

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

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

    AdjPWSLSLRepPcts[1] := 0;
    AdjPWSLSLRepPcts[2] := 0;
    AdjPWSLSLRepPcts[3] := 0;
    AdjPWSLSLRepPcts[4] := 0;
    AdjPWSLSLRepPcts[5] := 0;

    AdjPWSLSLRepPcts[1] := BaseCostSteps.Variables[BaseCostSteps.fI.pp_lslr_lsl] /

(BaseCostSteps.Variables[BaseCostSteps.fI.pp_lslr_lsl] +

BaseCostSteps.Variables[BaseCostSteps.fI.pp_lslr_partial]);

    AdjPWSLSLRepPcts[2] := BaseCostSteps.Variables[BaseCostSteps.fI.pp_lslr_partial]
/

(BaseCostSteps.Variables[BaseCostSteps.fI.pp_lslr_lsl] +

BaseCostSteps.Variables[BaseCostSteps.fI.pp_lslr_partial]);
  end;

  if Config.RunOptionOnly or Config.RunDifference then
  begin
    SetCCTCostEquationsData;
    ScenCostSteps.strLeadConcentrations := strLeadConcentrationsS;
    ScenCostSteps.SetRandomYears;
    ScenCostSteps.SetVariablesAndCalculate(CostingData, 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,
SchoolSampDataRec,
                                           pws_state);

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

```
      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;

    AdjPWSLSLRepPcts[1] := 0;
    AdjPWSLSLRepPcts[2] := 0;
    AdjPWSLSLRepPcts[3] := 0;
    AdjPWSLSLRepPcts[4] := 0;
    AdjPWSLSLRepPcts[5] := 0;

    if Config.OptionName = 'LCRR' then
    begin
      AdjPWSLSLRepPcts[1] := ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl]);

      AdjPWSLSLRepPcts[2] :=
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl]);

      AdjPWSLSLRepPcts[3] :=
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +
```

```
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl]);
    end
    else if Config.OptionName = 'LCRI' then
    begin
      if Config.LSLOption = 1 then
      begin
        ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_leadcon] := 0;
        ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon] := 0;
      end
      else
      if Config.LSLOption = 2 then
        ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon] := 0;

      AdjPWSLSLRepPcts[1] := ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_leadcon] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon]);

      AdjPWSLSLRepPcts[2] :=
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_leadcon] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon]);

      AdjPWSLSLRepPcts[3] :=
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_leadcon] +
```

```
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon]);

      AdjPWSLSLRepPcts[4] :=
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_leadcon] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_leadcon] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon]);

      AdjPWSLSLRepPcts[5] :=
ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon] /

(ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_partial] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_leadcon] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_gal_prev_lsl] +

ScenCostSteps.Variables[ScenCostSteps.fI.pp_lslr_galprev_leadcon]);
    end;
  end;

  LSLReplacement := 0;
  LSLReplacementMandatory := 0;
  LSLReplacementVoluntary := 0;
  LSLReplacementPopulation := 0;
  LSLReplacementPopMandatory := 0;
  LSLReplacementPopVoluntary := 0;
  LSLReplacementRequested := 0;
  LSLReplacementPopRequested := 0;

  AddModifyCCT := 0;
  AddModifyCCTPopulation := 0;

  // Total
  LSLReplaced := 0;
  LSLReplacedMandatory := 0;
  LSLReplacedVoluntary := 0;
  LSLReplacedRequested := 0;
  DeferralAny := 0;
```

```
DeferralPct := 0;
DeferralCap := 0;

PubNewPathO := 0;

LSLReplaceFullMandatory := 0;
LSLReplaceFullPopMandatory := 0;
LSLReplacePartialMandatory := 0;
LSLReplacePartialPopMandatory := 0;
LSLGalPrevMandatory := 0;
LSLGalPrevPopMandatory := 0;
LSLLeadConMandatory := 0;
LSLLeadConPopMandatory := 0;
LSLGalPrevLeadConMandatory := 0;
LSLGalPrevLeadConPopMandatory := 0;

LSLReplaceFullVoluntary := 0;
LSLReplaceFullPopVoluntary := 0;
LSLReplacePartialVoluntary := 0;
LSLReplacePartialPopVoluntary := 0;
LSLGalPrevVoluntary := 0;
LSLGalPrevPopVoluntary := 0;
LSLLeadConVoluntary := 0;
LSLLeadConPopVoluntary := 0;
LSLGalPrevLeadConVoluntary := 0;
LSLGalPrevLeadConPopVoluntary := 0;

// Non-State
LSLReplacedNS := 0;
LSLReplacedMandatoryNS := 0;
LSLReplacedVoluntaryNS := 0;

LSLReplaceFullMandatoryNS := 0;
LSLReplaceFullPopMandatoryNS := 0;
LSLReplacePartialMandatoryNS := 0;
LSLReplacePartialPopMandatoryNS := 0;
LSLGalPrevMandatoryNS := 0;
LSLGalPrevPopMandatoryNS := 0;
LSLLeadConMandatoryNS := 0;
LSLLeadConPopMandatoryNS := 0;
LSLGalPrevLeadConMandatoryNS := 0;
LSLGalPrevLeadConPopMandatoryNS := 0;

LSLReplaceFullVoluntaryNS := 0;
LSLReplaceFullPopVoluntaryNS := 0;
LSLReplacePartialVoluntaryNS := 0;
LSLReplacePartialPopVoluntaryNS := 0;
LSLGalPrevVoluntaryNS := 0;
LSLGalPrevPopVoluntaryNS := 0;
LSLLeadConVoluntaryNS := 0;
```

```
  LSLLeadConPopVoluntaryNS := 0;
  LSLGalPrevLeadConVoluntaryNS := 0;
  LSLGalPrevLeadConPopVoluntaryNS := 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;
  ToBin1Count:=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
      ToBin1Count := BaseCostSteps.ToBin1Count;

      // Total LSLR
      total_replacements := BaseCostSteps.SumTotLSLReplaceMand +
BaseCostSteps.SumTotLSLReplaceVol +
                            BaseCostSteps.SumTotLSLPartialReplaceMand +
BaseCostSteps.SumTotLSLPartialReplaceVol +
                            BaseCostSteps.SumTotLSLGalPrevMand +
BaseCostSteps.SumTotLSLGalPrevVol +
```

```
                            BaseCostSteps.SumTotLSLLeadConMand +
BaseCostSteps.SumTotLSLLeadConVol +
                            BaseCostSteps.SumTotLSLGalPrevLeadConMand +
BaseCostSteps.SumTotLSLGalPrevLeadConVol +
                            BaseCostSteps.LSLRequested;

      total_replacements_mand := BaseCostSteps.SumTotLSLReplaceMand +
                            BaseCostSteps.SumTotLSLPartialReplaceMand +
                            BaseCostSteps.SumTotLSLGalPrevMand +
                            BaseCostSteps.SumTotLSLLeadConMand +
                            BaseCostSteps.SumTotLSLGalPrevLeadConMand;

      total_replacements_vol := BaseCostSteps.SumTotLSLReplaceVol +
                            BaseCostSteps.SumTotLSLPartialReplaceVol +
                            BaseCostSteps.SumTotLSLGalPrevVol +
                            BaseCostSteps.SumTotLSLLeadConVol +
                            BaseCostSteps.SumTotLSLGalPrevLeadConVol;

      if total_replacements > 0 then
      begin
        LSLReplacement := 1;  // systems
        LSLReplacementPopulation := round(total_replacements *

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

      if total_replacements_mand > 0 then
      begin
        LSLReplacementMandatory := 1; // systems
        LSLReplacementPopMandatory := round(total_replacements_mand *

(CostingData.Population/CostingData.Connections));

        LSLReplaceFullMandatory := BaseCostSteps.SumTotLSLReplaceMand; // lines
        LSLReplaceFullPopMandatory := round(BaseCostSteps.SumTotLSLReplaceMand *

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

      if total_replacements_vol > 0 then
      begin
        LSLReplacementVoluntary := 1; // systems
        LSLReplacementPopVoluntary := round(total_replacements_vol *

(CostingData.Population/CostingData.Connections));

        LSLReplaceFullVoluntary := BaseCostSteps.SumTotLSLReplaceVol; // lines
        LSLReplaceFullPopVoluntary := round(BaseCostSteps.SumTotLSLReplaceVol *

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

```
      end;

      if BaseCostSteps.SumTotLSLPartialReplaceMand > 0 then
      begin
        LSLReplacePartialMandatory := BaseCostSteps.SumTotLSLPartialReplaceMand; //
lines
        LSLReplacePartialPopMandatory :=
round(BaseCostSteps.SumTotLSLPartialReplaceMand *

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

      if BaseCostSteps.SumTotLSLPartialReplaceVol > 0 then
      begin
        LSLReplacePartialVoluntary := BaseCostSteps.SumTotLSLPartialReplaceVol; //
lines
        LSLReplacePartialPopVoluntary :=
round(BaseCostSteps.SumTotLSLPartialReplaceVol *

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

      if BaseCostSteps.SumTotLSLGalPrevMand > 0 then
      begin
        LSLGalPrevMandatory := BaseCostSteps.SumTotLSLGalPrevMand;
        LSLGalPrevPopMandatory := round(BaseCostSteps.SumTotLSLGalPrevMand *

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

      if BaseCostSteps.SumTotLSLGalPrevVol > 0 then
      begin
        LSLGalPrevVoluntary := BaseCostSteps.SumTotLSLGalPrevVol;
        LSLGalPrevPopVoluntary := round(BaseCostSteps.SumTotLSLGalPrevVol *

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

      if BaseCostSteps.SumTotLSLLeadConMand > 0 then
      begin
        LSLLeadConMandatory := BaseCostSteps.SumTotLSLLeadConMand;
        LSLLeadConPopMandatory := round(BaseCostSteps.SumTotLSLLeadConMand *

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

      if BaseCostSteps.SumTotLSLLeadConVol > 0 then
      begin
        LSLLeadConVoluntary := BaseCostSteps.SumTotLSLLeadConVol;
        LSLLeadConPopVoluntary := round(BaseCostSteps.SumTotLSLLeadConVol *
```

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

      if BaseCostSteps.SumTotLSLGalPrevLeadConMand > 0 then
      begin
        LSLGalPrevLeadConMandatory := BaseCostSteps.SumTotLSLGalPrevLeadConMand;
        LSLGalPrevLeadConPopMandatory :=
round(BaseCostSteps.SumTotLSLGalPrevLeadConMand *

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

      if BaseCostSteps.SumTotLSLGalPrevLeadConVol > 0 then
      begin
        LSLGalPrevLeadConVoluntary := BaseCostSteps.SumTotLSLGalPrevLeadConVol;
        LSLGalPrevLeadConPopVoluntary :=
round(BaseCostSteps.SumTotLSLGalPrevLeadConVol *

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

      if BaseCostSteps.LSLRequested > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := round(BaseCostSteps.LSLRequested *

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

      LSLReplaced := BaseCostSteps.SumTotLSLReplace + BaseCostSteps.LSLRequested;
      LSLReplacedMandatory := BaseCostSteps.SumTotLSLReplaceMand;
      LSLReplacedVoluntary := BaseCostSteps.SumTotLSLReplaceVol;
      LSLReplacedRequested := BaseCostSteps.LSLRequested;
      DeferralAny := BaseCostSteps.Deferral_any;
      DeferralPct := BaseCostSteps.Deferral_pct;
      DeferralCap := BaseCostSteps.Deferral_cap;

      PubNewPathO := BaseCostSteps.pubNewPath;

      // Non-State LSLR
      total_replacements := BaseCostSteps.SumFedLSLReplaceMand +
BaseCostSteps.SumFedLSLReplaceVol +
                            BaseCostSteps.SumFedLSLPartialReplaceMand +
BaseCostSteps.SumFedLSLPartialReplaceVol +
                            BaseCostSteps.SumFedLSLGalPrevMand +
BaseCostSteps.SumFedLSLGalPrevVol +
                            BaseCostSteps.SumFedLSLLeadConMand +
BaseCostSteps.SumFedLSLLeadConVol +
                            BaseCostSteps.SumFedLSLGalPrevLeadConMand +
```

```
BaseCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_mand := BaseCostSteps.SumFedLSLReplaceMand +
                           BaseCostSteps.SumFedLSLPartialReplaceMand +
                           BaseCostSteps.SumFedLSLGalPrevMand +
                           BaseCostSteps.SumFedLSLLeadConMand +
                           BaseCostSteps.SumFedLSLGalPrevLeadConMand;

      total_replacements_vol := BaseCostSteps.SumFedLSLReplaceVol +
                           BaseCostSteps.SumFedLSLPartialReplaceVol +
                           BaseCostSteps.SumFedLSLGalPrevVol +
                           BaseCostSteps.SumFedLSLLeadConVol +
                           BaseCostSteps.SumFedLSLGalPrevLeadConVol;

      if BaseCostSteps.SumFedLSLReplaceMand > 0 then
      begin
        LSLReplaceFullMandatoryNS := BaseCostSteps.SumFedLSLReplaceMand; // lines
        LSLReplaceFullPopMandatoryNS := round(BaseCostSteps.SumFedLSLReplaceMand *

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

      if BaseCostSteps.SumFedLSLReplaceVol > 0 then
      begin
        LSLReplaceFullVoluntaryNS := BaseCostSteps.SumFedLSLReplaceVol; // lines
        LSLReplaceFullPopVoluntaryNS := round(BaseCostSteps.SumFedLSLReplaceVol *

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

      if BaseCostSteps.SumFedLSLPartialReplaceMand > 0 then
      begin
        LSLReplacePartialMandatoryNS := BaseCostSteps.SumFedLSLPartialReplaceMand;
// lines
        LSLReplacePartialPopMandatoryNS :=
round(BaseCostSteps.SumFedLSLPartialReplaceMand *

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

      if BaseCostSteps.SumFedLSLPartialReplaceVol > 0 then
      begin
        LSLReplacePartialVoluntaryNS := BaseCostSteps.SumFedLSLPartialReplaceVol; //
lines
        LSLReplacePartialPopVoluntaryNS :=
round(BaseCostSteps.SumFedLSLPartialReplaceVol *

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

```
      if BaseCostSteps.SumFedLSLGalPrevMand > 0 then
      begin
        LSLGalPrevMandatoryNS := BaseCostSteps.SumFedLSLGalPrevMand;
        LSLGalPrevPopMandatoryNS := round(BaseCostSteps.SumFedLSLGalPrevMand *

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

      if BaseCostSteps.SumFedLSLGalPrevVol > 0 then
      begin
        LSLGalPrevVoluntaryNS := BaseCostSteps.SumFedLSLGalPrevVol;
        LSLGalPrevPopVoluntaryNS := round(BaseCostSteps.SumFedLSLGalPrevVol *

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

      if BaseCostSteps.SumFedLSLLeadConMand > 0 then
      begin
        LSLLeadConMandatoryNS := BaseCostSteps.SumFedLSLLeadConMand;
        LSLLeadConPopMandatoryNS := round(BaseCostSteps.SumFedLSLLeadConMand *

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

      if BaseCostSteps.SumFedLSLLeadConVol > 0 then
      begin
        LSLLeadConVoluntaryNS := BaseCostSteps.SumFedLSLLeadConVol;
        LSLLeadConPopVoluntaryNS := round(BaseCostSteps.SumFedLSLLeadConVol *

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

      if BaseCostSteps.SumFedLSLGalPrevLeadConMand > 0 then
      begin
        LSLGalPrevLeadConMandatoryNS := BaseCostSteps.SumFedLSLGalPrevLeadConMand;
        LSLGalPrevLeadConPopMandatoryNS :=
round(BaseCostSteps.SumFedLSLGalPrevLeadConMand *

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

      if BaseCostSteps.SumFedLSLGalPrevLeadConVol > 0 then
      begin
        LSLGalPrevLeadConVoluntaryNS := BaseCostSteps.SumFedLSLGalPrevLeadConVol;
        LSLGalPrevLeadConPopVoluntaryNS :=
round(BaseCostSteps.SumFedLSLGalPrevLeadConVol *

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

```
LSLReplacedNS := BaseCostSteps.SumFedLSLReplace  + BaseCostSteps.LSLRequested;
LSLReplacedMandatoryNS := BaseCostSteps.SumFedLSLReplaceMand;
LSLReplacedVoluntaryNS := BaseCostSteps.SumFedLSLReplaceVol;

// CCT

if baseCostSteps.HasCCTCost then
begin
  AddModifyCCT := 1;
  AddModifyCCTPopulation := CostingData.Population;
end;

if baseCostSteps.CCTInstalled then
begin
  CCTInstalled := 1;
  CCTInstalledPopulation := CostingData.Population;
end;

if BaseCostSteps.CCTAdjusted then
begin
  CCTAdjusted := 1;
  CCTAdjustedPopulation := CostingData.Population;

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

  if BaseCostSteps.CCTAdjusted_tle then
  begin
    CCTAdjusted_tle := 1;
    CCTAdjustedPopulation_tle := CostingData.Population;
  end;
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 // VLS
  // Total LSLR
  if BaseCostSteps.LSLReplacedVLS > 0 then
  begin
    LSLReplacement := 1;  // systems
    LSLReplacementPopulation := BaseCostSteps.LSLReplacedPopVLS;
  end;

  if BaseCostSteps.LSLReplacedMandatoryVLS > 0 then
  begin
    LSLReplacementMandatory := 1; // systems
    LSLReplacementPopMandatory := BaseCostSteps.LSLReplacedPopVLS;
  end;

  if BaseCostSteps.LSLReplacedVoluntaryVLS > 0 then
  begin
    LSLReplacementVoluntary := 1; // systems
    LSLReplacementPopVoluntary := BaseCostSteps.LSLReplacedPopVLS;
  end;

  LSLReplaceFullMandatory := BaseCostSteps.LSLReplacedMandatoryVLS; // lines
  LSLReplaceFullPopMandatory := BaseCostSteps.LSLReplacedPopMandatoryVLS;
```

```
      LSLReplaceFullVoluntary := BaseCostSteps.LSLReplacedVoluntaryVLS; // lines
      LSLReplaceFullPopVoluntary := BaseCostSteps.LSLReplacedPopVoluntaryVLS;

      LSLReplacePartialMandatory := BaseCostSteps.LSLReplacePartialMandatoryVLS; //
lines
      LSLReplacePartialPopMandatory :=
BaseCostSteps.LSLReplacePartialPopMandatoryVLS;

      LSLReplacePartialVoluntary := BaseCostSteps.LSLReplacePartialVoluntaryVLS; //
lines
      LSLReplacePartialPopVoluntary :=
BaseCostSteps.LSLReplacePartialPopVoluntaryVLS;

      LSLGalPrevMandatory := BaseCostSteps.LSLGalPrevMandatoryVLS;
      LSLGalPrevPopMandatory := BaseCostSteps.LSLGalPrevPopMandatoryVLS;

      LSLGalPrevVoluntary := BaseCostSteps.LSLGalPrevVoluntaryVLS;
      LSLGalPrevPopVoluntary := BaseCostSteps.LSLGalPrevPopVoluntaryVLS;

      LSLLeadConMandatory := BaseCostSteps.LSLLeadConMandatoryVLS;
      LSLLeadConPopMandatory := BaseCostSteps.LSLLeadConPopMandatoryVLS;

      LSLLeadConVoluntary := BaseCostSteps.LSLLeadConVoluntaryVLS;
      LSLLeadConPopVoluntary := BaseCostSteps.LSLLeadConPopVoluntaryVLS;

      LSLGalPrevLeadConMandatory := BaseCostSteps.LSLGalPrevLeadConMandatoryVLS;
      LSLGalPrevLeadConPopMandatory :=
BaseCostSteps.LSLGalPrevLeadConPopMandatoryVLS;

      LSLGalPrevLeadConVoluntary := BaseCostSteps.LSLGalPrevLeadConVoluntaryVLS;
      LSLGalPrevLeadConPopVoluntary :=
BaseCostSteps.LSLGalPrevLeadConPopVoluntaryVLS;

      if BaseCostSteps.LSLRequestedVLS > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := BaseCostSteps.LSLRequestedPopVLS;
      end;

      LSLReplaced := BaseCostSteps.SumTotLSLReplace + BaseCostSteps.LSLRequestedVLS;
      LSLReplacedMandatory := BaseCostSteps.LSLReplacedMandatoryVLS;
      LSLReplacedVoluntary := BaseCostSteps.LSLReplacedVoluntaryVLS;
      LSLReplacedRequested := BaseCostSteps.LSLRequestedVLS;
      DeferralAny := BaseCostSteps.Deferral_anyvls;
      DeferralPct := BaseCostSteps.Deferral_pctvls;
      DeferralCap := BaseCostSteps.Deferral_capvls;

      // Non-State LSLR

      LSLReplaceFullMandatoryNS := BaseCostSteps.LSLReplacedMandatoryNSVLS; // lines
```

```
      LSLReplaceFullPopMandatoryNS := BaseCostSteps.LSLReplaceFullPopMandatoryNSVLS;

      LSLReplaceFullVoluntaryNS := BaseCostSteps.LSLReplacedVoluntaryNSVLS; // lines
      LSLReplaceFullPopVoluntaryNS := BaseCostSteps.LSLReplaceFullPopVoluntaryNSVLS;

      LSLReplacePartialMandatoryNS := BaseCostSteps.LSLReplacePartialMandatoryNSVLS;
// lines
      LSLReplacePartialPopMandatoryNS :=
BaseCostSteps.LSLReplacePartialPopMandatoryNSVLS;

      LSLReplacePartialVoluntaryNS := BaseCostSteps.LSLReplacePartialVoluntaryNSVLS;
// lines
      LSLReplacePartialPopVoluntaryNS :=
BaseCostSteps.LSLReplacePartialPopVoluntaryNSVLS;

      LSLGalPrevMandatoryNS := BaseCostSteps.LSLGalPrevMandatoryNSVLS;
      LSLGalPrevPopMandatoryNS := BaseCostSteps.LSLGalPrevPopMandatoryNSVLS;

      LSLGalPrevVoluntaryNS := BaseCostSteps.LSLGalPrevVoluntaryNSVLS;
      LSLGalPrevPopVoluntaryNS := BaseCostSteps.LSLGalPrevPopVoluntaryNSVLS;

      LSLLeadConMandatoryNS := BaseCostSteps.LSLLeadConMandatoryNSVLS;
      LSLLeadConPopMandatoryNS := BaseCostSteps.LSLLeadConPopMandatoryNSVLS;

      LSLLeadConVoluntaryNS := BaseCostSteps.LSLLeadConVoluntaryNSVLS;
      LSLLeadConPopVoluntaryNS := BaseCostSteps.LSLLeadConPopVoluntaryNSVLS;

      LSLGalPrevLeadConMandatoryNS := BaseCostSteps.LSLGalPrevLeadConMandatoryNSVLS;
      LSLGalPrevLeadConPopMandatoryNS :=
BaseCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS;

      LSLGalPrevLeadConVoluntaryNS := BaseCostSteps.LSLGalPrevLeadConVoluntaryNSVLS;
      LSLGalPrevLeadConPopVoluntaryNS :=
BaseCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS;

      if BaseCostSteps.LSLRequestedNSVLS > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := BaseCostSteps.LSLRequestedPopNSVLS;
      end;

      LSLReplacedNS := BaseCostSteps.SumFedLSLReplace +
BaseCostSteps.LSLRequestedVLS;
      LSLReplacedMandatoryNS := BaseCostSteps.LSLReplacedMandatoryNSVLS;
      LSLReplacedVoluntaryNS := BaseCostSteps.LSLReplacedVoluntaryNSVLS;

      if baseCostSteps.CCTInstalledVLS then
      begin
        CCTInstalled := 1;
        CCTInstalledPopulation := BaseCostSteps.CCTInstalledPopVLS;
```

```
end;

if baseCostSteps.CCTAdjustedVLS then
begin
  CCTAdjusted := 1;
  CCTAdjustedPopulation := BaseCostSteps.CCTAdjustedPopVLS;

  if BaseCostSteps.CCTAdjusted_ale then
  begin
    CCTAdjusted_ale := 1;
    CCTAdjustedPopulation_ale := BaseCostSteps.CCTAdjustedPopVLS_ale;
  end;

  if BaseCostSteps.CCTAdjusted_tle then
  begin
    CCTAdjusted_tle := 1;
    CCTAdjustedPopulation_tle := BaseCostSteps.CCTAdjustedPopVLS_tle;
  end;
end;

if baseCostSteps.CCTExistingVLS 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
      ToBin1Count := ScenCostSteps.ToBin1Count;

      // Total LSLR
      total_replacements := ScenCostSteps.SumTotLSLReplaceMand +
ScenCostSteps.SumTotLSLReplaceVol +
                            ScenCostSteps.SumTotLSLPartialReplaceMand +
ScenCostSteps.SumTotLSLPartialReplaceVol +
                            ScenCostSteps.SumTotLSLGalPrevMand +
ScenCostSteps.SumTotLSLGalPrevVol +
                            ScenCostSteps.SumTotLSLLeadConMand +
ScenCostSteps.SumTotLSLLeadConVol +
                            ScenCostSteps.SumTotLSLGalPrevLeadConMand +
ScenCostSteps.SumTotLSLGalPrevLeadConVol +
                            ScenCostSteps.LSLRequested;

      total_replacements_mand := ScenCostSteps.SumTotLSLReplaceMand +
                            ScenCostSteps.SumTotLSLPartialReplaceMand +
                            ScenCostSteps.SumTotLSLGalPrevMand +
                            ScenCostSteps.SumTotLSLLeadConMand +
                            ScenCostSteps.SumTotLSLGalPrevLeadConMand;

      total_replacements_vol := ScenCostSteps.SumTotLSLReplaceVol +
                            ScenCostSteps.SumTotLSLPartialReplaceVol +
                            ScenCostSteps.SumTotLSLGalPrevVol +
                            ScenCostSteps.SumTotLSLLeadConVol +
                            ScenCostSteps.SumTotLSLGalPrevLeadConVol;

      if total_replacements > 0 then
      begin
        LSLReplacement := 1;  // systems
        LSLReplacementPopulation := round(total_replacements *
```

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

      if total_replacements_mand > 0 then
      begin
        LSLReplacementMandatory := 1; // systems
        LSLReplacementPopMandatory := round(total_replacements_mand *

(CostingData.Population/CostingData.Connections));

        LSLReplaceFullMandatory := ScenCostSteps.SumTotLSLReplaceMand; // lines
        LSLReplaceFullPopMandatory := round(ScenCostSteps.SumTotLSLReplaceMand *

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

      if total_replacements_vol > 0 then
      begin
        LSLReplacementVoluntary := 1; // systems
        LSLReplacementPopVoluntary := round(total_replacements_vol *

(CostingData.Population/CostingData.Connections));

        LSLReplaceFullVoluntary := ScenCostSteps.SumTotLSLReplaceVol; // lines
        LSLReplaceFullPopVoluntary := round(ScenCostSteps.SumTotLSLReplaceVol *

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

      if ScenCostSteps.SumTotLSLPartialReplaceMand > 0 then
      begin
        LSLReplacePartialMandatory := ScenCostSteps.SumTotLSLPartialReplaceMand; //
lines
        LSLReplacePartialPopMandatory :=
round(ScenCostSteps.SumTotLSLPartialReplaceMand *

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

      if ScenCostSteps.SumTotLSLPartialReplaceVol > 0 then
      begin
        LSLReplacePartialVoluntary := ScenCostSteps.SumTotLSLPartialReplaceVol; //
lines
        LSLReplacePartialPopVoluntary :=
round(ScenCostSteps.SumTotLSLPartialReplaceVol *

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

      if ScenCostSteps.SumTotLSLGalPrevMand > 0 then
```

```
      begin
        LSLGalPrevMandatory := ScenCostSteps.SumTotLSLGalPrevMand;
        LSLGalPrevPopMandatory := round(ScenCostSteps.SumTotLSLGalPrevMand *

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

      if ScenCostSteps.SumTotLSLGalPrevVol > 0 then
      begin
        LSLGalPrevVoluntary := ScenCostSteps.SumTotLSLGalPrevVol;
        LSLGalPrevPopVoluntary := round(ScenCostSteps.SumTotLSLGalPrevVol *

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

      if ScenCostSteps.SumTotLSLLeadConMand > 0 then
      begin
        LSLLeadConMandatory := ScenCostSteps.SumTotLSLLeadConMand;
        LSLLeadConPopMandatory := round(ScenCostSteps.SumTotLSLLeadConMand *

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

      if ScenCostSteps.SumTotLSLLeadConVol > 0 then
      begin
        LSLLeadConVoluntary := ScenCostSteps.SumTotLSLLeadConVol;
        LSLLeadConPopVoluntary := round(ScenCostSteps.SumTotLSLLeadConVol *

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

      if ScenCostSteps.SumTotLSLGalPrevLeadConMand > 0 then
      begin
        LSLGalPrevLeadConMandatory := ScenCostSteps.SumTotLSLGalPrevLeadConMand;
        LSLGalPrevLeadConPopMandatory :=
round(ScenCostSteps.SumTotLSLGalPrevLeadConMand *

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

      if ScenCostSteps.SumTotLSLGalPrevLeadConVol > 0 then
      begin
        LSLGalPrevLeadConVoluntary := ScenCostSteps.SumTotLSLGalPrevLeadConVol;
        LSLGalPrevLeadConPopVoluntary :=
round(ScenCostSteps.SumTotLSLGalPrevLeadConVol *

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

      if ScenCostSteps.LSLRequested > 0 then
```

```
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := round(ScenCostSteps.LSLRequested *

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

      LSLReplaced := ScenCostSteps.SumTotLSLReplace + ScenCostSteps.LSLRequested;
      LSLReplacedMandatory := ScenCostSteps.SumTotLSLReplaceMand;
      LSLReplacedVoluntary := ScenCostSteps.SumTotLSLReplaceVol;
      LSLReplacedRequested := ScenCostSteps.LSLRequested;
      DeferralAny := ScenCostSteps.Deferral_any;
      DeferralPct := ScenCostSteps.Deferral_pct;
      DeferralCap := ScenCostSteps.Deferral_cap;

      PubNewPathO := ScenCostSteps.pubNewPath;

      // Non-State LSLR
      total_replacements := ScenCostSteps.SumFedLSLReplaceMand +
ScenCostSteps.SumFedLSLReplaceVol +
                            ScenCostSteps.SumFedLSLPartialReplaceMand +
ScenCostSteps.SumFedLSLPartialReplaceVol +
                            ScenCostSteps.SumFedLSLGalPrevMand +
ScenCostSteps.SumFedLSLGalPrevVol +
                            ScenCostSteps.SumFedLSLLeadConMand +
ScenCostSteps.SumFedLSLLeadConVol +
                            ScenCostSteps.SumFedLSLGalPrevLeadConMand +
ScenCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_mand := ScenCostSteps.SumFedLSLReplaceMand +
                            ScenCostSteps.SumFedLSLPartialReplaceMand +
                            ScenCostSteps.SumFedLSLGalPrevMand +
                            ScenCostSteps.SumFedLSLLeadConMand +
                            ScenCostSteps.SumFedLSLGalPrevLeadConMand;

      total_replacements_vol := ScenCostSteps.SumFedLSLReplaceVol +
                            ScenCostSteps.SumFedLSLPartialReplaceVol +
                            ScenCostSteps.SumFedLSLGalPrevVol +
                            ScenCostSteps.SumFedLSLLeadConVol +
                            ScenCostSteps.SumFedLSLGalPrevLeadConVol;

      if ScenCostSteps.SumFedLSLReplaceMand > 0 then
      begin
        LSLReplaceFullMandatoryNS := ScenCostSteps.SumFedLSLReplaceMand; // lines
        LSLReplaceFullPopMandatoryNS := round(ScenCostSteps.SumFedLSLReplaceMand *

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

      if ScenCostSteps.SumFedLSLReplaceVol > 0 then
```

```
      begin
        LSLReplaceFullVoluntaryNS := ScenCostSteps.SumFedLSLReplaceVol; // lines
        LSLReplaceFullPopVoluntaryNS := round(ScenCostSteps.SumFedLSLReplaceVol *

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

      if ScenCostSteps.SumFedLSLPartialReplaceMand > 0 then
      begin
        LSLReplacePartialMandatoryNS := ScenCostSteps.SumFedLSLPartialReplaceMand;
// lines
        LSLReplacePartialPopMandatoryNS :=
round(ScenCostSteps.SumFedLSLPartialReplaceMand *

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

      if ScenCostSteps.SumFedLSLPartialReplaceVol > 0 then
      begin
        LSLReplacePartialVoluntaryNS := ScenCostSteps.SumFedLSLPartialReplaceVol; //
lines
        LSLReplacePartialPopVoluntaryNS :=
round(ScenCostSteps.SumFedLSLPartialReplaceVol *

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

      if ScenCostSteps.SumFedLSLGalPrevMand > 0 then
      begin
        LSLGalPrevMandatoryNS := ScenCostSteps.SumFedLSLGalPrevMand;
        LSLGalPrevPopMandatoryNS := round(ScenCostSteps.SumFedLSLGalPrevMand *

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

      if ScenCostSteps.SumFedLSLGalPrevVol > 0 then
      begin
        LSLGalPrevVoluntaryNS := ScenCostSteps.SumFedLSLGalPrevVol;
        LSLGalPrevPopVoluntaryNS := round(ScenCostSteps.SumFedLSLGalPrevVol *

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

      if ScenCostSteps.SumFedLSLLeadConMand > 0 then
      begin
        LSLLeadConMandatoryNS := ScenCostSteps.SumFedLSLLeadConMand;
        LSLLeadConPopMandatoryNS := round(ScenCostSteps.SumFedLSLLeadConMand *

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

```
      if ScenCostSteps.SumFedLSLLeadConVol > 0 then
      begin
        LSLLeadConVoluntaryNS := ScenCostSteps.SumFedLSLLeadConVol;
        LSLLeadConPopVoluntaryNS := round(ScenCostSteps.SumFedLSLLeadConVol *

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

      if ScenCostSteps.SumFedLSLGalPrevLeadConMand > 0 then
      begin
        LSLGalPrevLeadConMandatoryNS := ScenCostSteps.SumFedLSLGalPrevLeadConMand;
        LSLGalPrevLeadConPopMandatoryNS :=
round(ScenCostSteps.SumFedLSLGalPrevLeadConMand *

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

      if ScenCostSteps.SumFedLSLGalPrevLeadConVol > 0 then
      begin
        LSLGalPrevLeadConVoluntaryNS := ScenCostSteps.SumFedLSLGalPrevLeadConVol;
        LSLGalPrevLeadConPopVoluntaryNS :=
round(ScenCostSteps.SumFedLSLGalPrevLeadConVol *

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

      LSLReplacedNS := ScenCostSteps.SumFedLSLReplace + ScenCostSteps.LSLRequested;
      LSLReplacedMandatoryNS := ScenCostSteps.SumFedLSLReplaceMand;
      LSLReplacedVoluntaryNS := ScenCostSteps.SumFedLSLReplaceVol;

      // CCT

      if ScenCostSteps.HasCCTCost then
      begin
        AddModifyCCT := 1;
        AddModifyCCTPopulation := CostingData.Population;
      end;

      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 begin // VLS
      // Total LSLR

      if ScenCostSteps.LSLReplacedVLS > 0 then
      begin
        LSLReplacement := 1;  // systems
        LSLReplacementPopulation := ScenCostSteps.LSLReplacedPopVLS;
      end;

      if ScenCostSteps.LSLReplacedMandatoryVLS > 0 then
      begin
        LSLReplacementMandatory := 1; // systems
        LSLReplacementPopMandatory := ScenCostSteps.LSLReplacedPopVLS;
      end;

      if ScenCostSteps.LSLReplacedVoluntaryVLS > 0 then
      begin
        LSLReplacementVoluntary := 1; // systems
        LSLReplacementPopVoluntary := ScenCostSteps.LSLReplacedPopVLS;
        LSLReplaceFullVoluntary := ScenCostSteps.SumTotLSLReplaceVol; // lines
      end;

      LSLReplaceFullMandatory := ScenCostSteps.LSLReplacedMandatoryVLS; // lines
      LSLReplaceFullPopMandatory := round(ScenCostSteps.LSLReplacedMandatoryVLS *

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialMandatory := ScenCostSteps.LSLReplacePartialMandatoryVLS; //
lines
      LSLReplacePartialPopMandatory :=
ScenCostSteps.LSLReplacePartialPopMandatoryVLS;

      LSLReplacePartialVoluntary := ScenCostSteps.LSLReplacePartialVoluntaryVLS; //
lines
      LSLReplacePartialPopVoluntary :=
ScenCostSteps.LSLReplacePartialPopVoluntaryVLS;

      LSLGalPrevMandatory := ScenCostSteps.LSLGalPrevMandatoryVLS;
      LSLGalPrevPopMandatory := ScenCostSteps.LSLGalPrevPopMandatoryVLS;

      LSLGalPrevVoluntary := ScenCostSteps.LSLGalPrevVoluntaryVLS;
      LSLGalPrevPopVoluntary := ScenCostSteps.LSLGalPrevPopVoluntaryVLS;

      LSLLeadConMandatory := ScenCostSteps.LSLLeadConMandatoryVLS;
      LSLLeadConPopMandatory := ScenCostSteps.LSLLeadConPopMandatoryVLS;
```

```
      LSLLeadConVoluntary := ScenCostSteps.LSLLeadConVoluntaryVLS;
      LSLLeadConPopVoluntary := ScenCostSteps.LSLLeadConPopVoluntaryVLS;

      LSLGalPrevLeadConMandatory := ScenCostSteps.LSLGalPrevLeadConMandatoryVLS;
      LSLGalPrevLeadConPopMandatory :=
ScenCostSteps.LSLGalPrevLeadConPopMandatoryVLS;

      LSLGalPrevLeadConVoluntary := ScenCostSteps.LSLGalPrevLeadConVoluntaryVLS;
      LSLGalPrevLeadConPopVoluntary :=
ScenCostSteps.LSLGalPrevLeadConPopVoluntaryVLS;

      if ScenCostSteps.LSLRequestedVLS > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := ScenCostSteps.LSLRequestedPopVLS;
      end;

      LSLReplaced := ScenCostSteps.SumTotLSLReplace + ScenCostSteps.LSLRequestedVLS;
      LSLReplacedMandatory := ScenCostSteps.LSLReplacedMandatoryVLS;
      LSLReplacedVoluntary := ScenCostSteps.LSLReplacedVoluntaryVLS;
      LSLReplacedRequested := ScenCostSteps.LSLRequestedVLS;
      DeferralAny := ScenCostSteps.Deferral_anyvls;
      DeferralPct := ScenCostSteps.Deferral_pctvls;
      DeferralCap := ScenCostSteps.Deferral_capvls;

      // Non-State LSLR

      if ScenCostSteps.LSLReplacedVoluntaryNSVLS > 0 then
      begin
        LSLReplacementVoluntary := 1; // systems
        LSLReplacementPopVoluntary := ScenCostSteps.LSLReplacedPopNSVLS;
      end;

      LSLReplaceFullMandatoryNS := ScenCostSteps.LSLReplacedMandatoryNSVLS; // lines
      LSLReplaceFullPopMandatoryNS := ScenCostSteps.LSLReplaceFullPopMandatoryNSVLS;

      LSLReplaceFullVoluntaryNS := ScenCostSteps.LSLReplacedVoluntaryNSVLS; // lines
      LSLReplaceFullPopVoluntaryNS := ScenCostSteps.LSLReplaceFullPopVoluntaryNSVLS;

      LSLReplacePartialMandatoryNS := ScenCostSteps.LSLReplacePartialMandatoryNSVLS;
// lines
      LSLReplacePartialPopMandatoryNS :=
ScenCostSteps.LSLReplacePartialPopMandatoryNSVLS;

      LSLReplacePartialVoluntaryNS := ScenCostSteps.LSLReplacePartialVoluntaryNSVLS;
// lines
      LSLReplacePartialPopVoluntaryNS :=
ScenCostSteps.LSLReplacePartialPopVoluntaryNSVLS;

      LSLGalPrevMandatoryNS := ScenCostSteps.LSLGalPrevMandatoryNSVLS;
```

```
      LSLGalPrevPopMandatoryNS := ScenCostSteps.LSLGalPrevPopMandatoryNSVLS;

      LSLGalPrevVoluntaryNS := ScenCostSteps.LSLGalPrevVoluntaryNSVLS;
      LSLGalPrevPopVoluntaryNS := ScenCostSteps.LSLGalPrevPopVoluntaryNSVLS;

      LSLLeadConMandatoryNS := ScenCostSteps.LSLLeadConMandatoryNSVLS;
      LSLLeadConPopMandatoryNS := ScenCostSteps.LSLLeadConPopMandatoryNSVLS;

      LSLLeadConVoluntaryNS := ScenCostSteps.LSLLeadConVoluntaryNSVLS;
      LSLLeadConPopVoluntaryNS := ScenCostSteps.LSLLeadConPopVoluntaryNSVLS;

      LSLGalPrevLeadConMandatoryNS := ScenCostSteps.LSLGalPrevLeadConMandatoryNSVLS;
      LSLGalPrevLeadConPopMandatoryNS :=
ScenCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS;

      LSLGalPrevLeadConVoluntaryNS := ScenCostSteps.LSLGalPrevLeadConVoluntaryNSVLS;
      LSLGalPrevLeadConPopVoluntaryNS :=
ScenCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS;

      if ScenCostSteps.LSLRequestedNSVLS > 0 then
      begin
        LSLReplacementRequested := 1;
        LSLReplacementPopRequested := ScenCostSteps.LSLRequestedPopNSVLS;
      end;

      LSLReplacedRequested := ScenCostSteps.LSLRequestedNSVLS;

      LSLReplacedNS := ScenCostSteps.SumFedLSLReplace +
ScenCostSteps.LSLRequestedVLS;
      LSLReplacedMandatoryNS := ScenCostSteps.LSLReplacedMandatoryNSVLS;
      LSLReplacedVoluntaryNS := ScenCostSteps.LSLReplacedVoluntaryNSVLS;

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

      if ScenCostSteps.CCTAdjustedVLS 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.CCTExistingVLS 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
      ToBin1Count := ScenCostSteps.ToBin1Count - BaseCostSteps.ToBin1Count;

      // Iotal LSLR
      total_replacements_Base := BaseCostSteps.SumTotLSLReplaceMand +
BaseCostSteps.SumTotLSLReplaceVol +
                           BaseCostSteps.SumTotLSLPartialReplaceMand +
BaseCostSteps.SumTotLSLPartialReplaceVol +
                           BaseCostSteps.SumTotLSLGalPrevMand +
BaseCostSteps.SumTotLSLGalPrevVol +
                           BaseCostSteps.SumTotLSLLeadConMand +
BaseCostSteps.SumTotLSLLeadConVol +
                           BaseCostSteps.SumTotLSLGalPrevLeadConMand +
BaseCostSteps.SumTotLSLGalPrevLeadConVol;

      total_replacements_Opt := ScenCostSteps.SumTotLSLReplaceMand +
ScenCostSteps.SumTotLSLReplaceVol +
                           ScenCostSteps.SumTotLSLPartialReplaceMand +
ScenCostSteps.SumTotLSLPartialReplaceVol +
                           ScenCostSteps.SumTotLSLGalPrevMand +
ScenCostSteps.SumTotLSLGalPrevVol +
                           ScenCostSteps.SumTotLSLLeadConMand +
ScenCostSteps.SumTotLSLLeadConVol +
                           ScenCostSteps.SumTotLSLGalPrevLeadConMand +
ScenCostSteps.SumTotLSLGalPrevLeadConVol;

      total_replacements_mand_Base := BaseCostSteps.SumTotLSLReplaceMand +
                           BaseCostSteps.SumTotLSLPartialReplaceMand +
                           BaseCostSteps.SumTotLSLGalPrevMand +
                           BaseCostSteps.SumTotLSLLeadConMand +
                           BaseCostSteps.SumTotLSLGalPrevLeadConMand;

      total_replacements_vol_Base := BaseCostSteps.SumTotLSLReplaceVol +
                           BaseCostSteps.SumTotLSLPartialReplaceVol +
                           BaseCostSteps.SumTotLSLGalPrevVol +
                           BaseCostSteps.SumTotLSLLeadConVol +
                           BaseCostSteps.SumTotLSLGalPrevLeadConVol;

      total_replacements_mand_Opt := ScenCostSteps.SumTotLSLReplaceMand +
                           ScenCostSteps.SumTotLSLPartialReplaceMand +
                           ScenCostSteps.SumTotLSLGalPrevMand +
                           ScenCostSteps.SumTotLSLLeadConMand +
                           ScenCostSteps.SumTotLSLGalPrevLeadConMand;

      total_replacements_vol_Opt := ScenCostSteps.SumTotLSLReplaceVol +
                           ScenCostSteps.SumTotLSLPartialReplaceVol +
                           ScenCostSteps.SumTotLSLGalPrevVol +
                           ScenCostSteps.SumTotLSLLeadConVol +
                           ScenCostSteps.SumTotLSLGalPrevLeadConVol;
```

```
      BaseHasIt := 0;
      ScenHasIt := 0;
      if total_replacements_Base > 0 then BaseHasIt := 1;
      if total_replacements_Opt > 0 then ScenHasIt := 1;
      LSLReplacement := ScenHasIt - BaseHasIt;

      LSLReplacementPopulation := round(total_replacements_Opt *

(CostingData.Population/CostingData.Connections));
      LSLReplacementPopulation := LSLReplacementPopulation -
round(total_replacements_Base *

(CostingData.Population/CostingData.Connections));

      BaseHasIt := 0;
      ScenHasIt := 0;
      if total_replacements_mand_Base > 0 then BaseHasIt := 1;
      if total_replacements_mand_Opt > 0 then ScenHasIt := 1;
      LSLReplacementMandatory := ScenHasIt - BaseHasIt;

//      if total_replacements_mand_Opt > 0 then
        LSLReplacementPopMandatory := round(total_replacements_mand_Opt *

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

(CostingData.Population/CostingData.Connections));

      BaseHasIt := 0;
      ScenHasIt := 0;
      if total_replacements_vol_Base > 0 then BaseHasIt := 1;
      if total_replacements_vol_Opt > 0 then ScenHasIt := 1;
      LSLReplacementVoluntary := ScenHasIt - BaseHasIt;

      LSLReplacementPopVoluntary := round(total_replacements_vol_Opt *

(CostingData.Population/CostingData.Connections));
      LSLReplacementPopVoluntary := LSLReplacementPopVoluntary -
round(total_replacements_vol_Base *

(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;
```

```
      LSLReplacementPopRequested := round(ScenCostSteps.LSLRequested *

(CostingData.Population/CostingData.Connections));
      LSLReplacementPopRequested := LSLReplacementPopRequested -
round(BaseCostSteps.LSLRequested *

(CostingData.Population/CostingData.Connections));

      LSLReplaceFullMandatory := ScenCostSteps.SumTotLSLReplaceMand -
BaseCostSteps.SumTotLSLReplaceMand;
      if ScenCostSteps.SumTotLSLReplaceMand > 0 then
        LSLReplaceFullPopMandatory := round(ScenCostSteps.SumTotLSLReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplaceFullVoluntary := ScenCostSteps.SumTotLSLReplaceVol -
BaseCostSteps.SumTotLSLReplaceVol;
      if ScenCostSteps.SumTotLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntary := round(ScenCostSteps.SumTotLSLReplaceVol *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.SumTotLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntary := LSLReplaceFullPopVoluntary -
round(BaseCostSteps.SumTotLSLReplaceMand *

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialMandatory := ScenCostSteps.SumTotLSLPartialReplaceMand -
BaseCostSteps.SumTotLSLPartialReplaceMand;
      if ScenCostSteps.SumTotLSLPartialReplaceMand > 0 then
        LSLReplacePartialPopMandatory :=
round(ScenCostSteps.SumTotLSLPartialReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialVoluntary := ScenCostSteps.SumTotLSLPartialReplaceVol -
BaseCostSteps.SumTotLSLPartialReplaceVol;
      if ScenCostSteps.SumTotLSLPartialReplaceVol > 0 then
        LSLReplacePartialPopVoluntary :=
```

```
round(ScenCostSteps.SumTotLSLPartialReplaceVol *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevMandatory := ScenCostSteps.SumTotLSLGalPrevMand -
BaseCostSteps.SumTotLSLGalPrevMand;
      if ScenCostSteps.SumTotLSLGalPrevMand > 0 then
        LSLGalPrevPopMandatory := round(ScenCostSteps.SumTotLSLGalPrevMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevVoluntary := ScenCostSteps.SumTotLSLGalPrevVol -
BaseCostSteps.SumTotLSLGalPrevVol;
      if ScenCostSteps.SumTotLSLGalPrevVol > 0 then
        LSLGalPrevPopVoluntary := round(ScenCostSteps.SumTotLSLGalPrevVol *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConMandatory := ScenCostSteps.SumTotLSLLeadConMand -
BaseCostSteps.SumTotLSLLeadConMand;
      if ScenCostSteps.SumTotLSLLeadConMand > 0 then
        LSLLeadConPopMandatory := round(ScenCostSteps.SumTotLSLLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConVoluntary := ScenCostSteps.SumTotLSLLeadConVol -
BaseCostSteps.SumTotLSLLeadConVol;
      if ScenCostSteps.SumTotLSLLeadConVol > 0 then
        LSLLeadConPopVoluntary := round(ScenCostSteps.SumTotLSLLeadConVol *
```

```
(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.SumTotLSLGalPrevVol > 0 then
        LSLLeadConPopVoluntary := LSLLeadConPopVoluntary -
round(BaseCostSteps.SumTotLSLLeadConVol *

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConMandatory := ScenCostSteps.SumTotLSLGalPrevLeadConMand -
BaseCostSteps.SumTotLSLGalPrevLeadConMand;
      if ScenCostSteps.SumTotLSLGalPrevLeadConMand > 0 then
        LSLGalPrevLeadConPopMandatory :=
round(ScenCostSteps.SumTotLSLGalPrevLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConVoluntary := ScenCostSteps.SumTotLSLGalPrevLeadConVol -
BaseCostSteps.SumTotLSLGalPrevLeadConVol;
      if ScenCostSteps.SumTotLSLGalPrevLeadConVol > 0 then
        LSLGalPrevLeadConPopVoluntary :=
round(ScenCostSteps.SumTotLSLGalPrevLeadConVol *

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

(CostingData.Population/CostingData.Connections));

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

      DeferralAny := ScenCostSteps.Deferral_any - BaseCostSteps.Deferral_any;
      DeferralPct := ScenCostSteps.Deferral_pct - BaseCostSteps.Deferral_any;
      DeferralCap := ScenCostSteps.Deferral_cap - BaseCostSteps.Deferral_any;

      // Non-State LSLR
      total_replacements_Base := BaseCostSteps.SumFedLSLReplaceMand +
BaseCostSteps.SumFedLSLReplaceVol +
                              BaseCostSteps.SumFedLSLPartialReplaceMand +
```

```
BaseCostSteps.SumFedLSLPartialReplaceVol +
                              BaseCostSteps.SumFedLSLGalPrevMand +
BaseCostSteps.SumFedLSLGalPrevVol +
                              BaseCostSteps.SumFedLSLLeadConMand +
BaseCostSteps.SumFedLSLLeadConVol +
                              BaseCostSteps.SumFedLSLGalPrevLeadConMand +
BaseCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_Opt := ScenCostSteps.SumFedLSLReplaceMand +
ScenCostSteps.SumFedLSLReplaceVol +
                              ScenCostSteps.SumFedLSLPartialReplaceMand +
ScenCostSteps.SumFedLSLPartialReplaceVol +
                              ScenCostSteps.SumFedLSLGalPrevMand +
ScenCostSteps.SumFedLSLGalPrevVol +
                              ScenCostSteps.SumFedLSLLeadConMand +
ScenCostSteps.SumFedLSLLeadConVol +
                              ScenCostSteps.SumFedLSLGalPrevLeadConMand +
ScenCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_mand_Base := BaseCostSteps.SumFedLSLReplaceMand +
                            BaseCostSteps.SumFedLSLPartialReplaceMand +
                            BaseCostSteps.SumFedLSLGalPrevMand +
                            BaseCostSteps.SumFedLSLLeadConMand +
                            BaseCostSteps.SumFedLSLGalPrevLeadConMand;

      total_replacements_vol_Base := BaseCostSteps.SumFedLSLReplaceVol +
                            BaseCostSteps.SumFedLSLPartialReplaceVol +
                            BaseCostSteps.SumFedLSLGalPrevVol +
                            BaseCostSteps.SumFedLSLLeadConVol +
                            BaseCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_mand_Opt := ScenCostSteps.SumFedLSLReplaceMand +
                            ScenCostSteps.SumFedLSLPartialReplaceMand +
                            ScenCostSteps.SumFedLSLGalPrevMand +
                            ScenCostSteps.SumFedLSLLeadConMand +
                            ScenCostSteps.SumFedLSLGalPrevLeadConMand;

      total_replacements_vol_Opt := ScenCostSteps.SumFedLSLReplaceVol +
                            ScenCostSteps.SumFedLSLPartialReplaceVol +
                            ScenCostSteps.SumFedLSLGalPrevVol +
                            ScenCostSteps.SumFedLSLLeadConVol +
                            ScenCostSteps.SumFedLSLGalPrevLeadConVol;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if total_replacements_vol_Base > 0 then BaseHasIt := 1;
      if total_replacements_vol_Opt > 0 then ScenHasIt := 1;
      LSLReplacementVoluntary := ScenHasIt - BaseHasIt;
      if total_replacements_vol_Opt > 0 then
        LSLReplacementPopVoluntary := round(total_replacements_vol_Opt *
```

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

(CostingData.Population/CostingData.Connections));

      LSLReplaceFullMandatoryNS := ScenCostSteps.SumFedLSLReplaceMand -
BaseCostSteps.SumFedLSLReplaceMand;
      if ScenCostSteps.SumFedLSLReplaceMand > 0 then
        LSLReplaceFullPopMandatoryNS := round(ScenCostSteps.SumFedLSLReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplaceFullVoluntaryNS := ScenCostSteps.SumFedLSLReplaceVol -
BaseCostSteps.SumFedLSLReplaceVol;
      if ScenCostSteps.SumFedLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntaryNS := round(ScenCostSteps.SumFedLSLReplaceVol *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.SumFedLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntaryNS := LSLReplaceFullPopVoluntaryNS -
round(BaseCostSteps.SumFedLSLReplaceMand *

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialMandatoryNS := ScenCostSteps.SumFedLSLPartialReplaceMand -
BaseCostSteps.SumFedLSLPartialReplaceMand;
      if ScenCostSteps.SumFedLSLPartialReplaceMand > 0 then
        LSLReplacePartialPopMandatoryNS :=
round(ScenCostSteps.SumFedLSLPartialReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialVoluntaryNS := ScenCostSteps.SumFedLSLPartialReplaceVol -
BaseCostSteps.SumFedLSLPartialReplaceVol;
      if ScenCostSteps.SumFedLSLPartialReplaceVol > 0 then
        LSLReplacePartialPopVoluntaryNS :=
round(ScenCostSteps.SumFedLSLPartialReplaceVol *
```

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevMandatoryNS := ScenCostSteps.SumFedLSLGalPrevMand -
BaseCostSteps.SumFedLSLGalPrevMand;
      if ScenCostSteps.SumFedLSLGalPrevMand > 0 then
        LSLGalPrevPopMandatoryNS := round(ScenCostSteps.SumFedLSLGalPrevMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevVoluntaryNS := ScenCostSteps.SumFedLSLGalPrevVol -
BaseCostSteps.SumFedLSLGalPrevVol;
      if ScenCostSteps.SumFedLSLGalPrevVol > 0 then
        LSLGalPrevPopVoluntaryNS := round(ScenCostSteps.SumFedLSLGalPrevVol *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConMandatoryNS := ScenCostSteps.SumFedLSLLeadConMand -
BaseCostSteps.SumFedLSLLeadConMand;
      if ScenCostSteps.SumFedLSLLeadConMand > 0 then
        LSLLeadConPopMandatoryNS := round(ScenCostSteps.SumFedLSLLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConVoluntaryNS := ScenCostSteps.SumFedLSLLeadConVol -
BaseCostSteps.SumFedLSLLeadConVol;
      if ScenCostSteps.SumFedLSLLeadConVol > 0 then
        LSLLeadConPopVoluntaryNS := round(ScenCostSteps.SumFedLSLLeadConVol *

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

```
      if BaseCostSteps.SumFedLSLGalPrevVol > 0 then
        LSLLeadConPopVoluntaryNS := LSLLeadConPopVoluntaryNS -
round(BaseCostSteps.SumFedLSLLeadConVol *

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConMandatoryNS := ScenCostSteps.SumFedLSLGalPrevLeadConMand -
BaseCostSteps.SumFedLSLGalPrevLeadConMand;
      if ScenCostSteps.SumFedLSLGalPrevLeadConMand > 0 then
        LSLGalPrevLeadConPopMandatoryNS :=
round(ScenCostSteps.SumFedLSLGalPrevLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConVoluntaryNS := ScenCostSteps.SumFedLSLGalPrevLeadConVol -
BaseCostSteps.SumFedLSLGalPrevLeadConVol;
      if ScenCostSteps.SumFedLSLGalPrevLeadConVol > 0 then
        LSLGalPrevLeadConPopVoluntaryNS :=
round(ScenCostSteps.SumFedLSLGalPrevLeadConVol *

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

(CostingData.Population/CostingData.Connections));

      LSLReplacedNS := (ScenCostSteps.SumFedLSLReplace + ScenCostSteps.LSLRequested)
-
                       (BaseCostSteps.SumFedLSLReplace +
ScenCostSteps.LSLRequested);
      LSLReplacedMandatoryNS := ScenCostSteps.SumFedLSLReplaceMand -
BaseCostSteps.SumFedLSLReplaceMand;
      LSLReplacedVoluntaryNS := ScenCostSteps.SumFedLSLReplaceVol -
BaseCostSteps.SumFedLSLReplaceVol;



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

```
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 // VLS
      // Total LSLR
      total_replacements_Base := BaseCostSteps.SumTotLSLReplaceMand +
BaseCostSteps.SumTotLSLReplaceVol +
                           BaseCostSteps.SumTotLSLPartialReplaceMand +
```

```
BaseCostSteps.SumTotLSLPartialReplaceVol +
                            BaseCostSteps.SumTotLSLGalPrevMand +
BaseCostSteps.SumTotLSLGalPrevVol +
                            BaseCostSteps.SumTotLSLLeadConMand +
BaseCostSteps.SumTotLSLLeadConVol +
                            BaseCostSteps.SumTotLSLGalPrevLeadConMand +
BaseCostSteps.SumTotLSLGalPrevLeadConVol;

      total_replacements_Opt := ScenCostSteps.SumTotLSLReplaceMand +
ScenCostSteps.SumTotLSLReplaceVol +
                            ScenCostSteps.SumTotLSLPartialReplaceMand +
ScenCostSteps.SumTotLSLPartialReplaceVol +
                            ScenCostSteps.SumTotLSLGalPrevMand +
ScenCostSteps.SumTotLSLGalPrevVol +
                            ScenCostSteps.SumTotLSLLeadConMand +
ScenCostSteps.SumTotLSLLeadConVol +
                            ScenCostSteps.SumTotLSLGalPrevLeadConMand +
ScenCostSteps.SumTotLSLGalPrevLeadConVol;

      total_replacements_mand_Base := BaseCostSteps.SumTotLSLReplaceMand +
                            BaseCostSteps.SumTotLSLPartialReplaceMand +
                            BaseCostSteps.SumTotLSLGalPrevMand +
                            BaseCostSteps.SumTotLSLLeadConMand +
                            BaseCostSteps.SumTotLSLGalPrevLeadConMand;

      total_replacements_mand_Opt := ScenCostSteps.SumTotLSLReplaceMand +
                            ScenCostSteps.SumTotLSLPartialReplaceMand +
                            ScenCostSteps.SumTotLSLGalPrevMand +
                            ScenCostSteps.SumTotLSLLeadConMand +
                            ScenCostSteps.SumTotLSLGalPrevLeadConMand;

      total_replacements_vol_Base := BaseCostSteps.SumTotLSLReplaceVol +
                            BaseCostSteps.SumTotLSLPartialReplaceVol +
                            BaseCostSteps.SumTotLSLGalPrevVol +
                            BaseCostSteps.SumTotLSLLeadConVol +
                            BaseCostSteps.SumTotLSLGalPrevLeadConVol;

      total_replacements_vol_Opt := ScenCostSteps.SumTotLSLReplaceVol +
                            ScenCostSteps.SumTotLSLPartialReplaceVol +
                            ScenCostSteps.SumTotLSLGalPrevVol +
                            ScenCostSteps.SumTotLSLLeadConVol +
                            ScenCostSteps.SumTotLSLGalPrevLeadConVol;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if total_replacements_Base > 0 then BaseHasIt := 1;
      if total_replacements_Opt > 0 then ScenHasIt := 1;

      LSLReplacement := ScenHasIt - BaseHasIt;
```

```
      if total_replacements_Opt > 0 then
        LSLReplacementPopulation := round(total_replacements_Opt *
ScenCostSteps.LSLReplacedPopVLS);
      if total_replacements_Base > 0 then
        LSLReplacementPopulation := LSLReplacementPopulation -
round(total_replacements_Base *
                                      ScenCostSteps.LSLReplacedPopVLS);

      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);

      LSLReplaceFullMandatory := ScenCostSteps.SumTotLSLReplaceMand -
BaseCostSteps.SumTotLSLReplaceMand;
      if ScenCostSteps.SumTotLSLReplaceMand > 0 then
        LSLReplaceFullPopMandatory := round(ScenCostSteps.SumTotLSLReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplaceFullVoluntary := ScenCostSteps.SumTotLSLReplaceVol -
BaseCostSteps.SumTotLSLReplaceVol;
      if ScenCostSteps.SumTotLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntary := round(ScenCostSteps.SumTotLSLReplaceVol *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.SumTotLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntary := LSLReplaceFullPopVoluntary -
round(BaseCostSteps.SumTotLSLReplaceMand *

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialMandatory := ScenCostSteps.SumTotLSLPartialReplaceMand -
BaseCostSteps.SumTotLSLPartialReplaceMand;
```

```
      if ScenCostSteps.SumTotLSLPartialReplaceMand > 0 then
        LSLReplacePartialPopMandatory :=
round(ScenCostSteps.SumTotLSLPartialReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialVoluntary := ScenCostSteps.SumTotLSLPartialReplaceVol -
BaseCostSteps.SumTotLSLPartialReplaceVol;
      if ScenCostSteps.SumTotLSLPartialReplaceVol > 0 then
        LSLReplacePartialPopVoluntary :=
round(ScenCostSteps.SumTotLSLPartialReplaceVol *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevMandatory := ScenCostSteps.SumTotLSLGalPrevMand -
BaseCostSteps.SumTotLSLGalPrevMand;
      if ScenCostSteps.SumTotLSLGalPrevMand > 0 then
        LSLGalPrevPopMandatory := round(ScenCostSteps.SumTotLSLGalPrevMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevVoluntary := ScenCostSteps.SumTotLSLGalPrevVol -
BaseCostSteps.SumTotLSLGalPrevVol;
      if ScenCostSteps.SumTotLSLGalPrevVol > 0 then
        LSLGalPrevPopVoluntary := round(ScenCostSteps.SumTotLSLGalPrevVol *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConMandatory := ScenCostSteps.SumTotLSLLeadConMand -
BaseCostSteps.SumTotLSLLeadConMand;
```

```
      if ScenCostSteps.SumTotLSLLeadConMand > 0 then
        LSLLeadConPopMandatory := round(ScenCostSteps.SumTotLSLLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConVoluntary := ScenCostSteps.SumTotLSLLeadConVol -
BaseCostSteps.SumTotLSLLeadConVol;
      if ScenCostSteps.SumTotLSLLeadConVol > 0 then
        LSLLeadConPopVoluntary := round(ScenCostSteps.SumTotLSLLeadConVol *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.SumTotLSLGalPrevVol > 0 then
        LSLLeadConPopVoluntary := LSLLeadConPopVoluntary -
round(BaseCostSteps.SumTotLSLLeadConVol *

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConMandatory := ScenCostSteps.SumTotLSLGalPrevLeadConMand -
BaseCostSteps.SumTotLSLGalPrevLeadConMand;
      if ScenCostSteps.SumTotLSLGalPrevLeadConMand > 0 then
        LSLGalPrevLeadConPopMandatory :=
round(ScenCostSteps.SumTotLSLGalPrevLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConVoluntary := ScenCostSteps.SumTotLSLGalPrevLeadConVol -
BaseCostSteps.SumTotLSLGalPrevLeadConVol;
      if ScenCostSteps.SumTotLSLGalPrevLeadConVol > 0 then
        LSLGalPrevLeadConPopVoluntary :=
round(ScenCostSteps.SumTotLSLGalPrevLeadConVol *

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

(CostingData.Population/CostingData.Connections));

      // Non-State LSLR
      total_replacements_Base := BaseCostSteps.SumFedLSLReplaceMand +
```

```
BaseCostSteps.SumFedLSLReplaceVol +
                           BaseCostSteps.SumFedLSLPartialReplaceMand +
BaseCostSteps.SumFedLSLPartialReplaceVol +
                           BaseCostSteps.SumFedLSLGalPrevMand +
BaseCostSteps.SumFedLSLGalPrevVol +
                           BaseCostSteps.SumFedLSLLeadConMand +
BaseCostSteps.SumFedLSLLeadConVol +
                           BaseCostSteps.SumFedLSLGalPrevLeadConMand +
BaseCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_Opt := ScenCostSteps.SumFedLSLReplaceMand +
ScenCostSteps.SumFedLSLReplaceVol +
                           ScenCostSteps.SumFedLSLPartialReplaceMand +
ScenCostSteps.SumFedLSLPartialReplaceVol +
                           ScenCostSteps.SumFedLSLGalPrevMand +
ScenCostSteps.SumFedLSLGalPrevVol +
                           ScenCostSteps.SumFedLSLLeadConMand +
ScenCostSteps.SumFedLSLLeadConVol +
                           ScenCostSteps.SumFedLSLGalPrevLeadConMand +
ScenCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_mand_Base := BaseCostSteps.SumFedLSLReplaceMand +
                           BaseCostSteps.SumFedLSLPartialReplaceMand +
                           BaseCostSteps.SumFedLSLGalPrevMand +
                           BaseCostSteps.SumFedLSLLeadConMand +
                           BaseCostSteps.SumFedLSLGalPrevLeadConMand;

      total_replacements_mand_Opt := ScenCostSteps.SumFedLSLReplaceMand +
                           ScenCostSteps.SumFedLSLPartialReplaceMand +
                           ScenCostSteps.SumFedLSLGalPrevMand +
                           ScenCostSteps.SumFedLSLLeadConMand +
                           ScenCostSteps.SumFedLSLGalPrevLeadConMand;

      total_replacements_vol_Base := BaseCostSteps.SumFedLSLReplaceVol +
                           BaseCostSteps.SumFedLSLPartialReplaceVol +
                           BaseCostSteps.SumFedLSLGalPrevVol +
                           BaseCostSteps.SumFedLSLLeadConVol +
                           BaseCostSteps.SumFedLSLGalPrevLeadConVol;

      total_replacements_vol_Opt := ScenCostSteps.SumFedLSLReplaceVol +
                           ScenCostSteps.SumFedLSLPartialReplaceVol +
                           ScenCostSteps.SumFedLSLGalPrevVol +
                           ScenCostSteps.SumFedLSLLeadConVol +
                           ScenCostSteps.SumFedLSLGalPrevLeadConVol;

      BaseHasIt := 0;
      ScenHasIt := 0;
      if total_replacements_vol_Base > 0 then BaseHasIt := 1;
      if total_replacements_vol_Opt > 0 then ScenHasIt := 1;
      LSLReplacementVoluntary := ScenHasIt - BaseHasIt;
```

```
      if total_replacements_vol_Opt > 0 then
        LSLReplacementPopVoluntary := round(total_replacements_vol_Opt *

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

(CostingData.Population/CostingData.Connections));

      LSLReplaceFullMandatoryNS := ScenCostSteps.SumFedLSLReplaceMand -
BaseCostSteps.SumFedLSLReplaceMand;
      if ScenCostSteps.SumFedLSLReplaceMand> 0 then
        LSLReplaceFullPopMandatoryNS := round(ScenCostSteps.SumFedLSLReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplaceFullVoluntaryNS := ScenCostSteps.SumFedLSLReplaceVol -
BaseCostSteps.SumFedLSLReplaceVol;
      if ScenCostSteps.SumFedLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntaryNS := round(ScenCostSteps.SumFedLSLReplaceVol *

(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.SumFedLSLReplaceVol > 0 then
        LSLReplaceFullPopVoluntaryNS := LSLReplaceFullPopVoluntaryNS -
round(BaseCostSteps.SumFedLSLReplaceMand *

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialMandatoryNS := ScenCostSteps.SumFedLSLPartialReplaceMand -
BaseCostSteps.SumFedLSLPartialReplaceMand;
      if ScenCostSteps.SumFedLSLPartialReplaceMand > 0 then
        LSLReplacePartialMandatoryNS :=
round(ScenCostSteps.SumFedLSLPartialReplaceMand *

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

(CostingData.Population/CostingData.Connections));

      LSLReplacePartialVoluntaryNS := ScenCostSteps.SumFedLSLPartialReplaceVol -
BaseCostSteps.SumFedLSLPartialReplaceVol;
      if ScenCostSteps.SumFedLSLPartialReplaceVol > 0 then
```

```
        LSLReplacePartialVoluntaryNS :=
round(ScenCostSteps.SumFedLSLPartialReplaceVol *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevMandatoryNS := ScenCostSteps.SumFedLSLGalPrevMand -
BaseCostSteps.SumFedLSLGalPrevMand;
      if ScenCostSteps.SumFedLSLGalPrevMand > 0 then
        LSLGalPrevPopMandatoryNS := round(ScenCostSteps.SumFedLSLGalPrevMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevVoluntaryNS := ScenCostSteps.SumFedLSLGalPrevVol -
BaseCostSteps.SumFedLSLGalPrevVol;
      if ScenCostSteps.SumFedLSLGalPrevVol > 0 then
        LSLGalPrevPopVoluntaryNS := round(ScenCostSteps.SumFedLSLGalPrevVol *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConMandatoryNS := ScenCostSteps.SumFedLSLLeadConMand -
BaseCostSteps.SumFedLSLLeadConMand;
      if ScenCostSteps.SumFedLSLLeadConMand > 0 then
        LSLLeadConPopMandatoryNS := round(ScenCostSteps.SumFedLSLLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLLeadConVoluntaryNS := ScenCostSteps.SumFedLSLLeadConVol -
BaseCostSteps.SumFedLSLLeadConVol;
      if ScenCostSteps.SumFedLSLLeadConVol > 0 then
        LSLLeadConPopVoluntaryNS := round(ScenCostSteps.SumFedLSLLeadConVol *
```

```
(CostingData.Population/CostingData.Connections));
      if BaseCostSteps.SumFedLSLGalPrevVol > 0 then
        LSLLeadConPopVoluntaryNS := LSLLeadConPopVoluntaryNS -
round(BaseCostSteps.SumFedLSLLeadConVol *

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConMandatoryNS := ScenCostSteps.SumFedLSLGalPrevLeadConMand -
BaseCostSteps.SumFedLSLGalPrevLeadConMand;
      if ScenCostSteps.SumFedLSLGalPrevLeadConMand > 0 then
        LSLGalPrevLeadConPopMandatoryNS :=
round(ScenCostSteps.SumFedLSLGalPrevLeadConMand *

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

(CostingData.Population/CostingData.Connections));

      LSLGalPrevLeadConVoluntaryNS := ScenCostSteps.SumFedLSLGalPrevLeadConVol -
BaseCostSteps.SumFedLSLGalPrevLeadConVol;
      if ScenCostSteps.SumFedLSLGalPrevLeadConVol > 0 then
        LSLGalPrevLeadConPopVoluntaryNS :=
round(ScenCostSteps.SumFedLSLGalPrevLeadConVol *

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

(CostingData.Population/CostingData.Connections));

      LSLReplacedNS := ScenCostSteps.LSLReplacedVLS - BaseCostSteps.LSLReplacedVLS;
      LSLReplacedMandatoryNS := ScenCostSteps.LSLReplacedMandatoryVLS -
BaseCostSteps.LSLReplacedMandatoryVLS;
      LSLReplacedVoluntaryNS := ScenCostSteps.LSLReplacedVoluntaryVLS -
BaseCostSteps.LSLReplacedVoluntaryVLS;



      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;

      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;
  vlsLSL: integer;
  pwsid: string;
  RecNo: integer;

  SLVlsLSLs: TStringList;
  SLVlsLSLs2: TStringList;

  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.LSLReplaceFullMandatoryVLS := 0;
    BaseCostSteps.LSLReplaceFullPopMandatoryVLS := 0;
    BaseCostSteps.LSLReplacePartialMandatoryVLS := 0;
    BaseCostSteps.LSLReplacePartialPopMandatoryVLS := 0;
    BaseCostSteps.LSLGalPrevMandatoryVLS := 0;
    BaseCostSteps.LSLGalPrevPopMandatoryVLS := 0;
    BaseCostSteps.LSLLeadConMandatoryVLS := 0;
    BaseCostSteps.LSLLeadConPopMandatoryVLS := 0;
    BaseCostSteps.LSLGalPrevLeadConMandatoryVLS := 0;
    BaseCostSteps.LSLGalPrevLeadConPopMandatoryVLS := 0;

    BaseCostSteps.LSLReplaceFullVoluntaryVLS := 0;
    BaseCostSteps.LSLReplaceFullPopVoluntaryVLS := 0;
    BaseCostSteps.LSLReplacePartialVoluntaryVLS := 0;
    BaseCostSteps.LSLReplacePartialPopVoluntaryVLS := 0;
    BaseCostSteps.LSLGalPrevVoluntaryVLS := 0;
    BaseCostSteps.LSLGalPrevPopVoluntaryVLS := 0;
    BaseCostSteps.LSLLeadConVoluntaryVLS := 0;
```

```
BaseCostSteps.LSLLeadConPopVoluntaryVLS := 0;
BaseCostSteps.LSLGalPrevLeadConVoluntaryVLS := 0;
BaseCostSteps.LSLGalPrevLeadConPopVoluntaryVLS := 0;

BaseCostSteps.LSLReplacedNSVLS := 0;
BaseCostSteps.LSLReplacedPopNSVLS := 0;
BaseCostSteps.LSLReplacedMandatoryNSVLS := 0;
BaseCostSteps.LSLReplacedVoluntaryNSVLS := 0;
BaseCostSteps.LSLReplacedPopMandatoryNSVLS := 0;
BaseCostSteps.LSLReplacedPopVoluntaryNSVLS := 0;
BaseCostSteps.LSLRequestedNSVLS := 0;
BaseCostSteps.LSLRequestedPopNSVLS := 0;

BaseCostSteps.LSLReplaceFullMandatoryNSVLS := 0;
BaseCostSteps.LSLReplaceFullPopMandatoryNSVLS := 0;
BaseCostSteps.LSLReplacePartialMandatoryNSVLS := 0;
BaseCostSteps.LSLReplacePartialPopMandatoryNSVLS := 0;
BaseCostSteps.LSLGalPrevMandatoryNSVLS := 0;
BaseCostSteps.LSLGalPrevPopMandatoryNSVLS := 0;
BaseCostSteps.LSLLeadConMandatoryNSVLS := 0;
BaseCostSteps.LSLLeadConPopMandatoryNSVLS := 0;
BaseCostSteps.LSLGalPrevLeadConMandatoryNSVLS := 0;
BaseCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS := 0;

BaseCostSteps.LSLReplaceFullVoluntaryNSVLS := 0;
BaseCostSteps.LSLReplaceFullPopVoluntaryNSVLS := 0;
BaseCostSteps.LSLReplacePartialVoluntaryNSVLS := 0;
BaseCostSteps.LSLReplacePartialPopVoluntaryNSVLS := 0;
BaseCostSteps.LSLGalPrevVoluntaryNSVLS := 0;
BaseCostSteps.LSLGalPrevPopVoluntaryNSVLS := 0;
BaseCostSteps.LSLLeadConVoluntaryNSVLS := 0;
BaseCostSteps.LSLLeadConPopVoluntaryNSVLS := 0;
BaseCostSteps.LSLGalPrevLeadConVoluntaryNSVLS := 0;
BaseCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS := 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;

  BaseCostSteps.deferral_anyvls := 0;
  BaseCostSteps.deferral_pctvls := 0;
  BaseCostSteps.deferral_capvls := 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.LSLReplaceFullMandatoryVLS := 0;
  ScenCostSteps.LSLReplaceFullPopMandatoryVLS := 0;
  ScenCostSteps.LSLReplacePartialMandatoryVLS := 0;
  ScenCostSteps.LSLReplacePartialPopMandatoryVLS := 0;
  ScenCostSteps.LSLGalPrevMandatoryVLS := 0;
  ScenCostSteps.LSLGalPrevPopMandatoryVLS := 0;
  ScenCostSteps.LSLLeadConMandatoryVLS := 0;
  ScenCostSteps.LSLLeadConPopMandatoryVLS := 0;
  ScenCostSteps.LSLGalPrevLeadConMandatoryVLS := 0;
  ScenCostSteps.LSLGalPrevLeadConPopMandatoryVLS := 0;

  ScenCostSteps.LSLReplaceFullVoluntaryVLS := 0;
  ScenCostSteps.LSLReplaceFullPopVoluntaryVLS := 0;
```

```
ScenCostSteps.LSLReplacePartialVoluntaryVLS := 0;
ScenCostSteps.LSLReplacePartialPopVoluntaryVLS := 0;
ScenCostSteps.LSLGalPrevVoluntaryVLS := 0;
ScenCostSteps.LSLGalPrevPopVoluntaryVLS := 0;
ScenCostSteps.LSLLeadConVoluntaryVLS := 0;
ScenCostSteps.LSLLeadConPopVoluntaryVLS := 0;
ScenCostSteps.LSLGalPrevLeadConVoluntaryVLS := 0;
ScenCostSteps.LSLGalPrevLeadConPopVoluntaryVLS := 0;

ScenCostSteps.LSLReplacedNSVLS := 0;
ScenCostSteps.LSLReplacedPopNSVLS := 0;
ScenCostSteps.LSLReplacedMandatoryNSVLS := 0;
ScenCostSteps.LSLReplacedVoluntaryNSVLS := 0;
ScenCostSteps.LSLReplacedPopMandatoryNSVLS := 0;
ScenCostSteps.LSLReplacedPopVoluntaryNSVLS := 0;
ScenCostSteps.LSLRequestedNSVLS := 0;
ScenCostSteps.LSLRequestedPopNSVLS := 0;

ScenCostSteps.LSLReplaceFullMandatoryNSVLS := 0;
ScenCostSteps.LSLReplaceFullPopMandatoryNSVLS := 0;
ScenCostSteps.LSLReplacePartialMandatoryNSVLS := 0;
ScenCostSteps.LSLReplacePartialPopMandatoryNSVLS := 0;
ScenCostSteps.LSLGalPrevMandatoryNSVLS := 0;
ScenCostSteps.LSLGalPrevPopMandatoryNSVLS := 0;
ScenCostSteps.LSLLeadConMandatoryNSVLS := 0;
ScenCostSteps.LSLLeadConPopMandatoryNSVLS := 0;
ScenCostSteps.LSLGalPrevLeadConMandatoryNSVLS := 0;
ScenCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS := 0;

ScenCostSteps.LSLReplaceFullVoluntaryNSVLS := 0;
ScenCostSteps.LSLReplaceFullPopVoluntaryNSVLS := 0;
ScenCostSteps.LSLReplacePartialVoluntaryNSVLS := 0;
ScenCostSteps.LSLReplacePartialPopVoluntaryNSVLS := 0;
ScenCostSteps.LSLGalPrevVoluntaryNSVLS := 0;
ScenCostSteps.LSLGalPrevPopVoluntaryNSVLS := 0;
ScenCostSteps.LSLLeadConVoluntaryNSVLS := 0;
ScenCostSteps.LSLLeadConPopVoluntaryNSVLS := 0;
ScenCostSteps.LSLGalPrevLeadConVoluntaryNSVLS := 0;
ScenCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS := 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;

  ScenCostSteps.deferral_anyvls := 0;
  ScenCostSteps.deferral_pctvls := 0;
  ScenCostSteps.deferral_capvls := 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_vls
  11 num_lsl_known_vls
```

```
    12 Num_lsl_unknown_lead_vls
    13 Num_lsl_unknown_vls
    14 Num_lsl_pws_vls
    15 NumberEPs
    16 pbasephpo4
    17 pbaseph
    18 pbasepo4
    19 Num_connect_vls
    20 Population
    21 Recno

 Read num_lsl_known_vls, num_lsl_unknown_vls, num_lsl_unknown_lead_vls, and
num_lsl_pws_vls from VLS EP spreadsheet

 }

  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;

      VLSEpWorkbook.NumberEPs := xls.GetCellValue(r, 15).AsVariant;
      csl(CostingData.pwsid + ' vlsLSL:' + vlsLSL.ToString + ' CostingData.LSL:' +
CostingData.LSL.ToString + ' VLSEpWorkbook.LSL:' + VLSEpWorkbook.LSL.ToString);

      if VarIsNumeric(xls.GetCellValue(r, 16).AsVariant) then
        VLSEpWorkbook.pbasephpo4 := xls.GetCellValue(r, 16).AsVariant
      else
        VLSEpWorkbook.pbasephpo4 := CostingData.CCTBoth ;

      if VarIsNumeric(xls.GetCellValue(r, 17).AsVariant) then
        VLSEpWorkbook.pbaseph := xls.GetCellValue(r, 17).AsVariant
      else
        VLSEpWorkbook.pbaseph := CostingData.CCTPH ;

      if VarIsNumeric(xls.GetCellValue(r, 18).AsVariant) then
        VLSEpWorkbook.pbasepo4 := xls.GetCellValue(r, 18).AsVariant
      else
        VLSEpWorkbook.pbasepo4 := CostingData.CCTPO4 ;

      VLSEpWorkbook.Population := xls.GetCellValue(r, 20).AsVariant;

      VLSEpWorkbook.LSL := xls.GetCellValue(r, 10).AsVariant;
      VLSEpWorkbook.num_lsl_known_vls := xls.GetCellValue(r, 11).AsVariant;
      VLSEpWorkbook.num_lsl_unknown_lead_vls := xls.GetCellValue(r, 12).AsVariant;
      VLSEpWorkbook.num_lsl_unknown_vls := xls.GetCellValue(r, 13).AsVariant;
      VLSEpWorkbook.num_lsl_pws_vls := xls.GetCellValue(r, 14).AsVariant;
      VLSEpWorkbook.Connections := xls.GetCellValue(r, 19).AsVariant;

      VLSEpWorkbook.RecNo := xls.GetCellValue(r, 21).AsVariant;

      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.SetVariablesAndCalculateVLS(CostingData, BAddCostingData,
VLSEpWorkbook, True, option,
                                                    CCTCostEquations,
                                                    CostingData.fBaseVars,
SchoolSampData,
                                                    Copy(CostingData.PWSId,1,2),
idcnt=1);

          BaseCostSteps.LSLReplacedVLS  := BaseCostSteps.LSLReplacedVLS +
BaseCostSteps.SumTotLSLReplace + BaseCostSteps.LSLRequested;
          BaseCostSteps.LSLReplacedPopVLS := BaseCostSteps.LSLReplacedPopVLS +

round((BaseCostSteps.SumTotLSLReplace + BaseCostSteps.LSLRequested) *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplacedMandatoryVLS  :=
BaseCostSteps.LSLReplacedMandatoryVLS + BaseCostSteps.SumTotLSLReplaceMand;
          BaseCostSteps.LSLReplacedVoluntaryVLS  :=
BaseCostSteps.LSLReplacedVoluntaryVLS + BaseCostSteps.SumTotLSLReplaceVol;

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

```
                round(BaseCostSteps.SumTotLSLReplaceVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLRequestedVLS := BaseCostSteps.LSLRequestedVLS +
BaseCostSteps.LSLRequested;
          BaseCostSteps.LSLRequestedPopVLS := BaseCostSteps.LSLRequestedPopVLS +
                round(BaseCostSteps.LSLRequested *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplaceFullMandatoryVLS :=
BaseCostSteps.LSLReplaceFullMandatoryVLS + BaseCostSteps.SumTotLSLReplaceMand;
          BaseCostSteps.LSLReplaceFullPopMandatoryVLS :=
BaseCostSteps.LSLReplaceFullPopMandatoryVLS +
                round(BaseCostSteps.SumTotLSLReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplacePartialMandatoryVLS :=
BaseCostSteps.LSLReplacePartialMandatoryVLS +
BaseCostSteps.SumTotLSLPartialReplaceMand;
          BaseCostSteps.LSLReplacePartialPopMandatoryVLS :=
BaseCostSteps.LSLReplacePartialPopMandatoryVLS +
                round(BaseCostSteps.SumTotLSLPartialReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevMandatoryVLS :=
BaseCostSteps.LSLGalPrevMandatoryVLS + BaseCostSteps.SumTotLSLGalPrevMand;
          BaseCostSteps.LSLGalPrevPopMandatoryVLS :=
BaseCostSteps.LSLGalPrevPopMandatoryVLS +
                round(BaseCostSteps.SumTotLSLGalPrevMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLLeadConMandatoryVLS :=
BaseCostSteps.LSLLeadConMandatoryVLS + BaseCostSteps.SumTotLSLLeadConMand;
          BaseCostSteps.LSLLeadConPopMandatoryVLS :=
BaseCostSteps.LSLLeadConPopMandatoryVLS +
                round(BaseCostSteps.SumTotLSLLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevLeadConMandatoryVLS :=
BaseCostSteps.LSLGalPrevLeadConMandatoryVLS +
BaseCostSteps.SumTotLSLGalPrevLeadConMand;
          BaseCostSteps.LSLGalPrevLeadConPopMandatoryVLS :=
BaseCostSteps.LSLGalPrevLeadConPopMandatoryVLS +
                round(BaseCostSteps.SumTotLSLGalPrevLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplacePartialVoluntaryVLS :=
BaseCostSteps.LSLReplacePartialVoluntaryVLS +
BaseCostSteps.SumTotLSLPartialReplaceVol;
          BaseCostSteps.LSLReplacePartialPopVoluntaryVLS :=
```

```
BaseCostSteps.LSLReplacePartialPopVoluntaryVLS +
                round(BaseCostSteps.SumTotLSLPartialReplaceVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevVoluntaryVLS :=
BaseCostSteps.LSLGalPrevVoluntaryVLS + BaseCostSteps.SumTotLSLGalPrevVol;
          BaseCostSteps.LSLGalPrevPopVoluntaryVLS :=
BaseCostSteps.LSLGalPrevPopVoluntaryVLS +
                round(BaseCostSteps.SumTotLSLGalPrevVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLLeadConVoluntaryVLS :=
BaseCostSteps.LSLLeadConVoluntaryVLS + BaseCostSteps.SumTotLSLLeadConVol;
          BaseCostSteps.LSLLeadConPopVoluntaryVLS :=
BaseCostSteps.LSLLeadConPopVoluntaryVLS +
                round(BaseCostSteps.SumTotLSLLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevLeadConVoluntaryVLS :=
BaseCostSteps.LSLGalPrevLeadConVoluntaryVLS +
BaseCostSteps.SumTotLSLGalPrevLeadConVol;
          BaseCostSteps.LSLGalPrevLeadConPopVoluntaryVLS :=
BaseCostSteps.LSLGalPrevLeadConPopVoluntaryVLS +
                round(BaseCostSteps.SumTotLSLGalPrevLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          if BaseCostSteps.deferral_any = 1 then BaseCostSteps.deferral_anyvls := 1;
          if BaseCostSteps.deferral_pct = 1 then BaseCostSteps.deferral_pctvls := 1;
          if BaseCostSteps.deferral_cap = 1 then BaseCostSteps.deferral_capvls := 1;

          // non-state
          BaseCostSteps.LSLReplacedNSVLS  := BaseCostSteps.LSLReplacedNSVLS +
BaseCostSteps.SumFedLSLReplace;
          BaseCostSteps.LSLReplacedPopNSVLS := BaseCostSteps.LSLReplacedPopNSVLS +

round(BaseCostSteps.SumFedLSLReplace *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplacedMandatoryNSVLS  :=
BaseCostSteps.LSLReplacedMandatoryNSVLS + BaseCostSteps.SumFedLSLReplaceMand;
          BaseCostSteps.LSLReplacedVoluntaryNSVLS  :=
BaseCostSteps.LSLReplacedVoluntaryNSVLS + BaseCostSteps.SumFedLSLReplaceVol;

          BaseCostSteps.LSLReplacedPopMandatoryNSVLS :=
BaseCostSteps.LSLReplacedPopMandatoryNSVLS +
                round(BaseCostSteps.SumFedLSLReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
          BaseCostSteps.LSLReplacedPopVoluntaryNSVLS :=
BaseCostSteps.LSLReplacedPopVoluntaryNSVLS +
                round(BaseCostSteps.SumFedLSLReplaceVol *
```

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

          BaseCostSteps.LSLReplaceFullMandatoryNSVLS :=
BaseCostSteps.LSLReplaceFullMandatoryNSVLS + BaseCostSteps.SumFedLSLReplaceMand;
          BaseCostSteps.LSLReplaceFullPopMandatoryNSVLS :=
BaseCostSteps.LSLReplaceFullPopMandatoryNSVLS +
                round(BaseCostSteps.SumFedLSLReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplacePartialMandatoryNSVLS :=
BaseCostSteps.LSLReplacePartialMandatoryNSVLS +
BaseCostSteps.SumFedLSLPartialReplaceMand;
          BaseCostSteps.LSLReplacePartialPopMandatoryNSVLS :=
BaseCostSteps.LSLReplacePartialPopMandatoryNSVLS +
                round(BaseCostSteps.SumFedLSLPartialReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevMandatoryNSVLS :=
BaseCostSteps.LSLGalPrevMandatoryNSVLS + BaseCostSteps.SumFedLSLGalPrevMand;
          BaseCostSteps.LSLGalPrevPopMandatoryNSVLS :=
BaseCostSteps.LSLGalPrevPopMandatoryNSVLS +
                round(BaseCostSteps.SumFedLSLGalPrevMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLLeadConMandatoryNSVLS :=
BaseCostSteps.LSLLeadConMandatoryNSVLS + BaseCostSteps.SumFedLSLLeadConMand;
          BaseCostSteps.LSLLeadConPopMandatoryNSVLS :=
BaseCostSteps.LSLLeadConPopMandatoryNSVLS +
                round(BaseCostSteps.SumFedLSLLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevLeadConMandatoryNSVLS :=
BaseCostSteps.LSLGalPrevLeadConMandatoryNSVLS +
BaseCostSteps.SumFedLSLGalPrevLeadConMand;
          BaseCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS :=
BaseCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS +
                round(BaseCostSteps.SumFedLSLGalPrevLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLReplacePartialVoluntaryNSVLS :=
BaseCostSteps.LSLReplacePartialVoluntaryNSVLS +
BaseCostSteps.SumFedLSLPartialReplaceVol;
          BaseCostSteps.LSLReplacePartialPopVoluntaryNSVLS :=
BaseCostSteps.LSLReplacePartialPopVoluntaryNSVLS +
                round(BaseCostSteps.SumFedLSLPartialReplaceVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevVoluntaryNSVLS :=
BaseCostSteps.LSLGalPrevVoluntaryNSVLS + BaseCostSteps.SumFedLSLGalPrevVol;
          BaseCostSteps.LSLGalPrevPopVoluntaryNSVLS :=
```

```
BaseCostSteps.LSLGalPrevPopVoluntaryNSVLS +
                round(BaseCostSteps.SumFedLSLGalPrevVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLLeadConVoluntaryNSVLS :=
BaseCostSteps.LSLLeadConVoluntaryNSVLS + BaseCostSteps.SumFedLSLLeadConVol;
          BaseCostSteps.LSLLeadConPopVoluntaryNSVLS :=
BaseCostSteps.LSLLeadConPopVoluntaryNSVLS +
                round(BaseCostSteps.SumFedLSLLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          BaseCostSteps.LSLGalPrevLeadConVoluntaryNSVLS :=
BaseCostSteps.LSLGalPrevLeadConVoluntaryNSVLS +
BaseCostSteps.SumFedLSLGalPrevLeadConVol;
          BaseCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS :=
BaseCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS +
                round(BaseCostSteps.SumFedLSLGalPrevLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

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

          if BaseCostSteps.CCTAdjusted then
          begin
            BaseCostSteps.CCTAdjustedVLS := true;
            BaseCostSteps.CCTAdjustedPopVLS := BaseCostSteps.CCTAdjustedPopVLS +
VLSEpWorkbook.Population;

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

          if BaseCostSteps.CCTExisting then
          begin
            BaseCostSteps.CCTExistingVLS := true;
            BaseCostSteps.CCTExistingPopVLS := BaseCostSteps.CCTExistingPopVLS +
```

```
VLSEpWorkbook.Population;
          end;

          if BaseCostSteps.HasFindAndFixCost then
          begin
            BaseCostSteps.HasFindAndFixCostVLS := 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;
      end else begin
        ScenCostSteps.SetVariablesAndCalculateVLS(CostingData,
SAddCostingData,VLSEpWorkbook, True, option,
                                                  CCTCostEquations,
                                                  CostingData.fScenVars,
SchoolSampData,
                                                  Copy(CostingData.PWSId,1,2),
idcnt=1);

          ScenCostSteps.LSLReplacedVLS  := ScenCostSteps.LSLReplacedVLS +
ScenCostSteps.SumTotLSLReplace + ScenCostSteps.LSLRequested;
          ScenCostSteps.LSLReplacedPopVLS := ScenCostSteps.LSLReplacedPopVLS +

round((ScenCostSteps.SumTotLSLReplace + ScenCostSteps.LSLRequested) *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplacedMandatoryVLS  :=
ScenCostSteps.LSLReplacedMandatoryVLS + ScenCostSteps.SumTotLSLReplaceMand;
          ScenCostSteps.LSLReplacedVoluntaryVLS  :=
ScenCostSteps.LSLReplacedVoluntaryVLS + ScenCostSteps.SumTotLSLReplaceVol;

          ScenCostSteps.LSLReplacedPopMandatoryVLS :=
ScenCostSteps.LSLReplacedPopMandatoryVLS +
                 round(ScenCostSteps.SumTotLSLReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
          ScenCostSteps.LSLReplacedPopVoluntaryVLS :=
ScenCostSteps.LSLReplacedPopVoluntaryVLS +
                 round(ScenCostSteps.SumTotLSLReplaceVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLRequestedVLS := ScenCostSteps.LSLRequestedVLS +
ScenCostSteps.LSLRequested;
          ScenCostSteps.LSLRequestedPopVLS := ScenCostSteps.LSLRequestedPopVLS +
                 round(ScenCostSteps.LSLRequested *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplaceFullMandatoryVLS :=
ScenCostSteps.LSLReplaceFullMandatoryVLS + ScenCostSteps.SumTotLSLReplaceMand;
          ScenCostSteps.LSLReplaceFullPopMandatoryVLS :=
ScenCostSteps.LSLReplaceFullPopMandatoryVLS +
                 round(ScenCostSteps.SumTotLSLReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplacePartialMandatoryVLS :=
ScenCostSteps.LSLReplacePartialMandatoryVLS +
```

```
ScenCostSteps.SumTotLSLPartialReplaceMand;
          ScenCostSteps.LSLReplacePartialPopMandatoryVLS :=
ScenCostSteps.LSLReplacePartialPopMandatoryVLS +
                round(ScenCostSteps.SumTotLSLPartialReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLGalPrevMandatoryVLS :=
ScenCostSteps.LSLGalPrevMandatoryVLS + ScenCostSteps.SumTotLSLGalPrevMand;
          ScenCostSteps.LSLGalPrevPopMandatoryVLS :=
ScenCostSteps.LSLGalPrevPopMandatoryVLS +
                round(ScenCostSteps.SumTotLSLGalPrevMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLLeadConMandatoryVLS :=
ScenCostSteps.LSLLeadConMandatoryVLS + ScenCostSteps.SumTotLSLLeadConMand;
          ScenCostSteps.LSLLeadConPopMandatoryVLS :=
ScenCostSteps.LSLLeadConPopMandatoryVLS +
                round(ScenCostSteps.SumTotLSLLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLGalPrevLeadConMandatoryVLS :=
ScenCostSteps.LSLGalPrevLeadConMandatoryVLS +
ScenCostSteps.SumTotLSLGalPrevLeadConMand;
          ScenCostSteps.LSLGalPrevLeadConPopMandatoryVLS :=
ScenCostSteps.LSLGalPrevLeadConPopMandatoryVLS +
                round(ScenCostSteps.SumTotLSLGalPrevLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplacePartialVoluntaryVLS :=
ScenCostSteps.LSLReplacePartialVoluntaryVLS +
ScenCostSteps.SumTotLSLPartialReplaceVol;
          ScenCostSteps.LSLReplacePartialPopVoluntaryVLS :=
ScenCostSteps.LSLReplacePartialPopVoluntaryVLS +
                round(ScenCostSteps.SumTotLSLPartialReplaceVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLGalPrevVoluntaryVLS :=
ScenCostSteps.LSLGalPrevVoluntaryVLS + ScenCostSteps.SumTotLSLGalPrevVol;
          ScenCostSteps.LSLGalPrevPopVoluntaryVLS :=
ScenCostSteps.LSLGalPrevPopVoluntaryVLS +
                round(ScenCostSteps.SumTotLSLGalPrevVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLLeadConVoluntaryVLS :=
ScenCostSteps.LSLLeadConVoluntaryVLS + ScenCostSteps.SumTotLSLLeadConVol;
          ScenCostSteps.LSLLeadConPopVoluntaryVLS :=
ScenCostSteps.LSLLeadConPopVoluntaryVLS +
                round(ScenCostSteps.SumTotLSLLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
```

```
          ScenCostSteps.LSLGalPrevLeadConVoluntaryVLS :=
ScenCostSteps.LSLGalPrevLeadConVoluntaryVLS +
ScenCostSteps.SumTotLSLGalPrevLeadConVol;
          ScenCostSteps.LSLGalPrevLeadConPopVoluntaryVLS :=
ScenCostSteps.LSLGalPrevLeadConPopVoluntaryVLS +
                round(ScenCostSteps.SumTotLSLGalPrevLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          if ScenCostSteps.deferral_any = 1 then ScenCostSteps.deferral_anyvls := 1;
          if ScenCostSteps.deferral_pct = 1 then ScenCostSteps.deferral_pctvls := 1;
          if ScenCostSteps.deferral_cap = 1 then ScenCostSteps.deferral_capvls := 1;

          // non-state
          ScenCostSteps.LSLReplacedNSVLS  := ScenCostSteps.LSLReplacedNSVLS +
ScenCostSteps.SumFedLSLReplace;
          ScenCostSteps.LSLReplacedPopNSVLS := ScenCostSteps.LSLReplacedPopNSVLS +

round(ScenCostSteps.SumFedLSLReplace *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplacedMandatoryNSVLS  :=
ScenCostSteps.LSLReplacedMandatoryNSVLS + ScenCostSteps.SumFedLSLReplaceMand;
          ScenCostSteps.LSLReplacedVoluntaryNSVLS  :=
ScenCostSteps.LSLReplacedVoluntaryNSVLS + ScenCostSteps.SumFedLSLReplaceVol;

          ScenCostSteps.LSLReplacedPopMandatoryNSVLS :=
ScenCostSteps.LSLReplacedPopMandatoryNSVLS +
                round(ScenCostSteps.SumFedLSLReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
          ScenCostSteps.LSLReplacedPopVoluntaryNSVLS :=
ScenCostSteps.LSLReplacedPopVoluntaryNSVLS +
                round(ScenCostSteps.SumFedLSLReplaceVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplaceFullMandatoryNSVLS :=
ScenCostSteps.LSLReplaceFullMandatoryNSVLS + ScenCostSteps.SumFedLSLReplaceMand;
          ScenCostSteps.LSLReplaceFullPopMandatoryNSVLS :=
ScenCostSteps.LSLReplaceFullPopMandatoryNSVLS +
                round(ScenCostSteps.SumFedLSLReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplacePartialMandatoryNSVLS :=
ScenCostSteps.LSLReplacePartialMandatoryNSVLS +
ScenCostSteps.SumFedLSLPartialReplaceMand;
          ScenCostSteps.LSLReplacePartialPopMandatoryNSVLS :=
ScenCostSteps.LSLReplacePartialPopMandatoryNSVLS +
                round(ScenCostSteps.SumFedLSLPartialReplaceMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLGalPrevMandatoryNSVLS :=
```

```
ScenCostSteps.LSLGalPrevMandatoryNSVLS + ScenCostSteps.SumFedLSLGalPrevMand;
          ScenCostSteps.LSLGalPrevPopMandatoryNSVLS :=
ScenCostSteps.LSLGalPrevPopMandatoryNSVLS +
                 round(ScenCostSteps.SumFedLSLGalPrevMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLLeadConMandatoryNSVLS :=
ScenCostSteps.LSLLeadConMandatoryNSVLS + ScenCostSteps.SumFedLSLLeadConMand;
          ScenCostSteps.LSLLeadConPopMandatoryNSVLS :=
ScenCostSteps.LSLLeadConPopMandatoryNSVLS +
                 round(ScenCostSteps.SumFedLSLLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLGalPrevLeadConMandatoryNSVLS :=
ScenCostSteps.LSLGalPrevLeadConMandatoryNSVLS +
ScenCostSteps.SumFedLSLGalPrevLeadConMand;
          ScenCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS :=
ScenCostSteps.LSLGalPrevLeadConPopMandatoryNSVLS +
                 round(ScenCostSteps.SumFedLSLGalPrevLeadConMand *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLReplacePartialVoluntaryNSVLS :=
ScenCostSteps.LSLReplacePartialVoluntaryNSVLS +
ScenCostSteps.SumFedLSLPartialReplaceVol;
          ScenCostSteps.LSLReplacePartialPopVoluntaryNSVLS :=
ScenCostSteps.LSLReplacePartialPopVoluntaryNSVLS +
                 round(ScenCostSteps.SumFedLSLPartialReplaceVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLGalPrevVoluntaryNSVLS :=
ScenCostSteps.LSLGalPrevVoluntaryNSVLS + ScenCostSteps.SumFedLSLGalPrevVol;
          ScenCostSteps.LSLGalPrevPopVoluntaryNSVLS :=
ScenCostSteps.LSLGalPrevPopVoluntaryNSVLS +
                 round(ScenCostSteps.SumFedLSLGalPrevVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLLeadConVoluntaryNSVLS :=
ScenCostSteps.LSLLeadConVoluntaryNSVLS + ScenCostSteps.SumFedLSLLeadConVol;
          ScenCostSteps.LSLLeadConPopVoluntaryNSVLS :=
ScenCostSteps.LSLLeadConPopVoluntaryNSVLS +
                 round(ScenCostSteps.SumFedLSLLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));

          ScenCostSteps.LSLGalPrevLeadConVoluntaryNSVLS :=
ScenCostSteps.LSLGalPrevLeadConVoluntaryNSVLS +
ScenCostSteps.SumFedLSLGalPrevLeadConVol;
          ScenCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS :=
ScenCostSteps.LSLGalPrevLeadConPopVoluntaryNSVLS +
                 round(ScenCostSteps.SumFedLSLGalPrevLeadConVol *
(VLSEpWorkbook.Population/VLSEpWorkbook.Connections));
```

```
          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;

            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;
          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;

{ TAggInputRec }
```

```
procedure TAggInputRec.SetDefault;
begin
  AggName := '';
  BaselineValue := 0;
  OptionValue := 0;
  Difference := 0;
  SIdx:=-1;
  SIdxp:=-1;
  BIdx:=-1;
  BIdxp:=-1;
end;

end.
```