

LCRMetricCollector.pas

```
unit LCRMetricCollector;

interface

uses SysUtils, Types, Classes, SafeWaterUncertBucket, MtxVec,
    Math387, Statistics, LCRGlobals, LCRResultsFile,
    LCRConfig, StrUtils,
    Agglomerators, Contrns, DB,
dialogs
    ;

const

    mtCost=0;
    mtBenefitDollars=1;
    mtBenefitCases=2;
    mtNetBenefits=3;
    mtBenefitCounts=4;
    mtInfo=9;
    mtUMRA=10;

type

    TMetricCounters=record
        TotalValues, NANINFVals : int64;
    end;

    TMetricStore=class
    private
        fNumberTrials : integer;
        fCurIndex : integer;
        tmpResults : TtmpMetrics;
        tmpPercVec : TVec;
        fPercInc : double;
        fSampleProb : double; //probability of sample = 1/NumSamples;
    public
        MetricDef : PMetricDef;
        Output : TMetricOutput;

        PValue, PProb : pdouble; //pointer to value and probability to collect
        YearVec : TVec; // pointer to yearly data if saved by year

        Counters : TMetricCounters;

        constructor Create(fConfig : TLCRConfig);
        destructor Destroy; override;
        procedure Init;
```

```

                                LCRMetricCollector.pas

procedure Reset;
procedure CollectWeightedObs;
procedure ContamLevelDone(const ContamLevel : integer);

procedure ApplyOccDist(const Conc : TVec; const Threshold : double);

end;

TMetricList=class
  ResultsFile : TLCRResultsFile;

  Metrics : TList;
  Config : TLCRConfig;
  constructor Create(fConfig : TLCRConfig);
  destructor Destroy; override;
  procedure AddOutputMetric(PValue,PProb: pdouble; YearVec: TVec; Name: string;
MetricType: byte; SaveVar,SaveYear,IgnoreZeros: boolean;
                                aOption: string; aDiscRate:single; SaveSingle:
boolean=false; UncOnly: boolean=False);
  procedure DoneAddingOutputs;

  procedure CollectWeightedObs;
  procedure ContamLevelDone(const ContamLevel : integer);
  procedure ApplyOccDist(const Conc : TVec; const Threshold : double);

  procedure Reset;

  {$ifdef debug}
  procedure ReportToStrings(T : TStrings; Kind : word);
  {$endif}
end;

TMetricSaver=class
  Mean          : double;
  UncMean       : array of double;

  SumMean       : double;
  UncSum        : array of double;

  QuantileMeans : array of double;
  UncQuantiles  : array of array of double;

  Yearly        : array of double;

  constructor create(aConfig : TLCRConfig; aMetricDef : PMetricDef);
  constructor createFromStream(Strm : TStream);
  procedure LoadFromStream(Strm : TStream);
  procedure SaveToStream(Strm : TStream);

```

```

LCRMetricCollector.pas

destructor Destroy; override;
end;

TCategoryMetrics=class
private
    fMetricList : TMetricList;
    fConfig : TLCRConfig;
    fN,fUncN : integer;
    fVarWght : double;
    tmpVarQuants : array of double;

public
    Name,Filter : string;
    Results : array of TMetricSaver; //one for every metric

    tmpYearly : array of TVec; //at most one for every metric
    UnctmpYearly : array of TVec; //at most one for every metric
    VariabilityQuants : array of TStreamQuantiles; //one for every metric
    UncertaintyQuants : array of array of TStreamQuantiles; //one for every metric -
each metric has a number of saved percentiles
    SumUncertaintyQuants : array of TStreamQuantiles; //sum of metric distribution
across unc
    MeanUncertainty : array of TStreamQuantiles; //one for every metric

    constructor Create(aConfig : TLCRConfig; aMetricList : TMetricList;
aName,aFilter : string);
    destructor Destroy; override;

    procedure CollectVariability(const wgt : double);
    procedure CollectUncertainty;

    procedure GenerateFinalOutput;

    procedure LoadFromStream(Strm : TStream);
    procedure SaveToStream(Strm : TStream);
    procedure AddCategoryMetrics(aCatMetrics : TCategoryMetrics); //method to add
individual metrics to master list

    procedure SaveResultsToStream(Strm : TStream);

    function MemSize : integer;
    function NANINFCount: integer;
    function AggPoints: int64;
end;

TCategoryList=class
private
    fMetricList : TMetricList;

```

```

LCRMetricCollector.pas

fConfig : TLCRConfig;
fFilename : string;
public
  Categories : TObjectList;
  constructor Create(aConfig : TLCRConfig; aMetricList : TMetricList; aFilename :
string);
  destructor Destroy; override;

  procedure CollectVariability(const Cat : integer; const Wgt : double);
  //TODO: Warning very ungeneral proc here....
  //should allow for a general concept of uncertainty only metrics
  procedure GenerateSpecialMetrics;
  procedure CollectUncertainty;

  procedure GenerateFinalOutput;
  procedure SaveCategoryOutput;

  procedure SaveToStream(AStream : TStream);
  procedure LoadFromStream(AStream : TStream);
  procedure AddCategoryList(ACat : TcategoryList);

  function MemSize : integer;
  function NANINFCount: integer;
  function AggPoints: int64;

  property FileName: string read fFilename;
end;

TCategoryOutputReader=class
private
  function StripIt(S : string) : string;
public
  //used to read final result file
  //saved via TCategoryList.SaveCategoryOutput;
  Metrics : array of TMetricDef;
  Results : array of TObjectList;
  Names,Filters : TStringList;
  QuantilePoints : TDoubleDynArray; //array of quantile points to store across
uncertainty
  UncQuantilePoints : TDoubleDynArray; //array of quantile points to output for
uncertainty

  constructor Create(aFileName : string);
  destructor Destroy; override;

```

```

                                LCRMetricCollector.pas
    procedure DumpToDataset(DS : TDataset; OptionName: string);
    procedure DumpToTabDelim(FN : string; Mean : boolean; NoICRCosts: boolean);
    procedure DumpMeanSumToDataset(DS : TDataset; TargMetricName : string; DoMean :
boolean);

    procedure Dump3VarToDataset(DS: TDataset; TargMetricName: string; s1,s2,s3 :
integer);

    end;

```

implementation

```
{ TMetricStore }
```

```

procedure TMetricStore.CollectWeightedObs;
var v : double;
begin
    inc(Counters.TotalValues);

    V:=PValue^;
    if (IsNan(V)) or (IsInf(V)) then begin
        inc(Counters.NANINFVals);
        exit;
    end;

    if PProb^=0 then
        exit;

    inc(fCurIndex);

    if fCurIndex>high(tmpResults) then
        raise Exception.create('Too many points in CollectWeightedObs');

    tmpResults[fCurIndex].Value:=PValue^;
    tmpResults[fCurIndex].Probability:=PProb^;
end;

constructor TMetricStore.create(fConfig : TLCRConfig);
begin
    inherited create;
    fCurIndex:=-1;
    fNumberTrials:=fConfig.NumberOfTrials;

```

```

end;

destructor TMetricStore.Destroy;
begin
  if MetricDef^.SaveVariability then begin
    tmpPercVec.Free;
  end;
  inherited;
end;

procedure TMetricStore.Init;
begin
  fPercInc:=1/PercentileValuesToStore;
  fSampleProb:=1/fNumberTrials;
  fillchar(Counters,SizeOF(Counters),0);

  if MetricDef^.SaveVariability then begin
    //ToDo must determine a good value for this
    SetLength(tmpResults,10000);
    tmpPercVec:=TVec.Create;
    tmpPercVec.Size(PercentileValuesToStore);
  end else begin
    SetLength(tmpResults,0);
  end;

  Reset;
end;

procedure TMetricStore.Reset;
begin
  fCurIndex:=-1;
end;

procedure TMetricStore.ContamLevelDone(const ContamLevel : integer);
var ProbSum : double;
    i : integer;
begin
  //try
  if MetricDef^.SingleValue then begin
    fCurIndex:=-1;
    exit;
  end;

  if MetricDef^.SaveVariability then begin
    if fCurIndex>-1 then begin
      //BSortTmpMetrics(@tmpResults,0,fCurIndex);
      QSortTmpMetrics(tmpResults,0,fCurIndex);
      GenVariabilityPercs(tmpResults,tmpPercVec,ProbSum,0,fCurIndex);
    end;
  end;
end;

```

```

                                LCRMetricCollector.pas
    end else begin
        //Case when all values are NAN,INF should be addresssed.....
        tmpPercVec.SetZero;
    end;
    Output.Results.SetCol(tmpPercVec,ContamLevel);
end else begin
    if MetricDef^.SaveByYear then begin
        Output.Results.SetCol(YearVec,ContamLevel);
    end else begin
        Output.Results.Values[0,ContamLevel]:=PValue^;
    end;
end;

fCurIndex:=-1;

end;

procedure TMetricStore.ApplyOccDist(const Conc: TVec; const Threshold : double);
begin
    if MetricDef.SingleValue then
        Output.CurMean:=PValue^
    else
        Output.ApplyDist(Conc,Threshold);
end;

{ TMetricList }

procedure TMetricList.AddOutputMetric(PValue,PProb: pdouble;
    YearVec: TVec; Name: string; MetricType: byte;
    SaveVar,SaveYear,IgnoreZeros: boolean;
    aOption: string; aDiscRate: single; SaveSingle: boolean=false; UncOnly:
    boolean=False);
var T : TMetricStore;
begin
    T:=TMetricStore.create(Config);

    T.MetricDef:=ResultsFile.AddMetricDef(Name,MetricType,SaveVar,SaveYear,IgnoreZeros,a
    Option,aDiscRate,SaveSingle,UncOnly);

    T.PValue:=PValue;
    T.PProb:=PProb;
    T.YearVec:=YearVec;
    Metrics.Add(T);
end;

procedure TMetricList.CollectWeightedObs;
var T : TMetricStore;
    i : integer;

```

LCRMetricCollector.pas

```
begin
  for i:=0 to Metrics.Count-1 do begin
    if TMetricStore(Metrics.Items[i]).MetricDef^.SaveVariability then
      TMetricStore(Metrics.Items[i]).CollectWeightedObs;
    end;
  end;

  constructor TMetricList.Create(fConfig : TLCRConfig);
  begin
    Metrics:=TList.Create;
    Config := fConfig;
  end;

  destructor TMetricList.Destroy;
  var i : integer;
  begin
    for i:=0 to Metrics.Count-1 do TMetricStore(Metrics.Items[i]).Free;
    Metrics.Free;
    inherited;
  end;

  procedure TMetricList.ContamLevelDone(const ContamLevel : integer);
  var T : TMetricStore;
      i : integer;
  begin
    for i:=0 to Metrics.Count-1 do begin
      T:=TMetricStore(Metrics.Items[i]);
      T.ContamLevelDone(ContamLevel);
    end;
  end;

  {$ifdef debug}
  procedure TMetricList.ReportToStrings(T: TStrings; Kind: word);
  var TOM : TMetricStore;
      i,x : integer;
  begin
    end;
  {$endif}

  procedure TMetricList.Reset;
  var i : integer;
  begin
    for i:=0 to Metrics.Count-1 do
      TMetricStore(Metrics.Items[i]).Reset;
    end;

  procedure TMetricList.DoneAddingOutputs;
  var i,ix : integer;
```


LCRMetricCollector.pas

```

    T : TMetricStore;
begin
    //Assign an output object to each metric
    for i:=0 to Metrics.Count-1 do begin
        T:=TMetricStore(Metrics.Items[i]);
        ix:=ResultsFile.MetricResults.IndexOf(String(T.MetricDef^.Name));
        if ix<0 then begin
            raise exception.create('Metric not found in DoneAddingOutputs');
        end;
        T.Output:=TMetricOutput(ResultsFile.MetricResults.Objects[ix]);
        T.Init;
    end;
end;

procedure TMetricList.ApplyOccDist(const Conc: TVec; const Threshold : double);
var i : integer;
    T : TMetricStore;
begin
    for i:=0 to Metrics.Count-1 do begin
        T:=TMetricStore(Metrics.Items[i]);
        T.ApplyOccDist(Conc,Threshold);
    end;
end;

{ TCategoryMetrics }

procedure TCategoryMetrics.AddCategoryMetrics(
    aCatMetrics: TCategoryMetrics);
var i,j : integer;
begin
    try
        inc(fUncN);
        for i:=low(UncertaintyQuants) to high(UncertaintyQuants) do begin
            for j:=low(fConfig.QuantilePoints) to high(fConfig.QuantilePoints) do begin
                UncertaintyQuants[i,j].AddQuantiles(aCatMetrics.UncertaintyQuants[i,j]);
            end;

            SumUncertaintyQuants[i].AddQuantiles(aCatMetrics.SumUncertaintyQuants[i]);

            if TMetricStore(fMetricList.Metrics.Items[i]).MetricDef.SaveByYear then begin
                UncTmpYearly[i].Add(aCatMetrics.UncTmpYearly[i]);
            end;

            MeanUncertainty[i].AddQuantiles(aCatMetrics.MeanUncertainty[i]);
        end;
    except

```

```

LCRMetricCollector.pas
    raise exception.Create('error in AddCat '+inttostr(i)+' '+inttostr(j));
end;
end;

function TCategoryMetrics.AggPoints: int64;
var i,j,total : integer;
begin
    total:=0;
    for i:= low(VariabilityQuants) to high(VariabilityQuants) do
        total:=total+TStreamQuantiles(VariabilityQuants[i]).NumPoints;
    for i:= low(SumUncertaintyQuants) to high(SumUncertaintyQuants) do
        total:=total+TStreamQuantiles(SumUncertaintyQuants[i]).NumPoints;
    for i:= low(MeanUncertainty) to high(MeanUncertainty) do
        total:=total+TStreamQuantiles(MeanUncertainty[i]).NumPoints;
    for i:= low(UncertaintyQuants) to high(UncertaintyQuants) do
        for j:= low(UncertaintyQuants[i]) to high(UncertaintyQuants[i]) do
            total:=total+TStreamQuantiles(UncertaintyQuants[i,j]).NumPoints;
    Result:=Total;
end;

procedure TCategoryMetrics.CollectUncertainty;
var i,j : integer;
begin
    try
        inc(fUncN);
        for i:=low(UncertaintyQuants) to high(UncertaintyQuants) do begin
            if not TMetricStore(fMetricList.Metrics.Items[i]).MetricDef.CategoryOnly
then begin
                for j:=low(fConfig.QuantilePoints) to high(fConfig.QuantilePoints) do
begin
                    UncertaintyQuants[i,j].AddPoint(VariabilityQuants[i].Quantile(fConfig.QuantilePoints
[j]),1);
                        end;

                    if TMetricStore(fMetricList.Metrics.Items[i]).MetricDef.SaveVariability
then
                        SumUncertaintyQuants[i].AddPoint(VariabilityQuants[i].Sum)
                    else
                        SumUncertaintyQuants[i].AddPoint(VariabilityQuants[i].Sum);

                    if TMetricStore(fMetricList.Metrics.Items[i]).MetricDef.SaveByYear then
begin
                        tmpYearly[i].Scale(1/fN);
                        UncTmpYearly[i].Add(tmpYearly[i]);
                    end;

```

```

                                LCRMetricCollector.pas
        MeanUncertainty[i].AddPoint(VariabilityQuants[i].Mean);
    end else begin
        //do nothing - taken care of in GeneratreSpecialMetrics...
    end;

    if tmpYearly[i]<>nil then tmpYearly[i].SetZero;
    VariabilityQuants[i].Reset;
end;
fN:=0;
except
    on e:exception do raise exception.Create('error in CollectUncertainty
'+inttostr(i)+' ','+inttostr(j)+' ,msg: '+e.Message);
end;
end;

procedure TCategoryMetrics.CollectVariability(const wgt : double);
var i,j : integer;
    TMD : PMetricDef;
    TMO : TMetricStore;
begin
    try
        inc(fN);
        for i:=low(VariabilityQuants) to high(VariabilityQuants) do begin
            TMO:=TMetricStore(fMetricList.Metrics.Items[i]);
            TMD:=TMO.MetricDef;
            if TMD.CategoryOnly then continue;
            if TMD.SaveVariability then begin
                for j:=0 to TMO.Output.CurPercs.Length-1 do begin
                    VariabilityQuants[i].AddPoint(TMO.Output.CurPercs.Values[j],fVarWght*wgt);
                end;
            end else
            if TMD.SaveByYear then begin
                for j:=0 to TMO.Output.CurYearly.Length-1 do

tmpYearly[i].Values[j]:=tmpYearly[i].Values[j]+TMO.Output.CurYearly.Values[j]*wgt;
                //Saving variabililty for final Year

            VariabilityQuants[i].AddPoint(TMO.Output.CurYearly.Values[TMO.Output.CurYearly.Lengt
h-1],wgt);
            end else begin
                VariabilityQuants[i].AddPoint(TMO.Output.CurMean,wgt);
            end;
        end;
    except
    on E:EXCEPTION do exception.create(E.MESSAGE+' in collectVariability');
end;
end;

```

```

LCRMetricCollector.pas
constructor TCategoryMetrics.create(aConfig: TLCRConfig;
  aMetricList: TMetricList; aName,aFilter : string);
var i,j : integer;
begin
  inherited create;
  fConfig:=aConfig;
  fMetricList:=aMetricList;
  fN:=0;
  fUncN:=0;
  Name:=aName;
  Filter:=aFilter;
  SetLength(VariabilityQuants,aMetricList.Metrics.Count);
  SetLength(UncertaintyQuants,aMetricList.Metrics.Count);
  SetLength(MeanUncertainty,aMetricList.Metrics.Count);
  SetLength(SumUncertaintyQuants,aMetricList.Metrics.Count);
  SetLength(Results,aMetricList.Metrics.Count);
  SetLength(tmpYearly,aMetricList.Metrics.Count);
  SetLength(UnctmpYearly,aMetricList.Metrics.Count);

  setlength(tmpVarQuants,high(aConfig.QuantilePoints)+1);
  for i:=0 to high(UncertaintyQuants) do
    SetLength(UncertaintyQuants[i],high(aConfig.QuantilePoints)+1);

  //create necessary objects
  for i:=low(VariabilityQuants) to high(VariabilityQuants) do begin
    VariabilityQuants[i]:=TStreamQuantiles.create(aConfig.QuantilePrecision);
    if TMetricStore(fMetricList.Metrics.Items[i]).MetricDef.IgnoreZeros then
      VariabilityQuants[i].IgnoreZero:=True;
    SumUncertaintyQuants[i]:=TStreamQuantiles.create(aConfig.QuantilePrecision);
    MeanUncertainty[i]:=TStreamQuantiles.create(aConfig.QuantilePrecision);

  Results[i]:=TMetricSaver.Create(aConfig,TMetricStore(fMetricList.Metrics.Items[i]).MetricDef);
    for j:=low(UncertaintyQuants[i]) to high(UncertaintyQuants[i]) do
      UncertaintyQuants[i,j]:=TStreamQuantiles.create(aConfig.QuantilePrecision);
    end;

  for i:=0 to fMetricList.Metrics.Count-1 do
    if TMetricStore(fMetricList.Metrics.Items[i]).MetricDef.SaveByYear then begin
      tmpYearly[i]:=TVec.Create;
      tmpYearly[i].Size(aConfig.YearsOfAnalysis);
      tmpYearly[i].Setzero;
      UnctmpYearly[i]:=TVec.Create;
      UnctmpYearly[i].Size(aConfig.YearsOfAnalysis);
      UnctmpYearly[i].Setzero;
    end else begin
      tmpYearly[i]:=nil;
      UnctmpYearly[i]:=nil;
    end;
  end;
end;

```

```

end;

fVarWght:=1/PercentileValuesToStore;
end;

destructor TCategoryMetrics.Destroy;
var i,j : integer;
begin
  for i:=low(VariabilityQuants) to high(VariabilityQuants) do begin
    VariabilityQuants[i].Free;
    SumUncertaintyQuants[i].Free;
    MeanUncertainty[i].Free;

    if tmpYearly[i]<>nil then tmpYearly[i].Free;
    if UnctmpYearly[i]<>nil then UnctmpYearly[i].Free;
    //TODO the following line was causing an intermitant Invalid pointer op.
    try
      Results[i].Free;
    except
      CSL('results free error:'+i.ToString);
    end;

    for j:=low(UncertaintyQuants[i]) to high(UncertaintyQuants[i]) do
      try
        UncertaintyQuants[i,j].Free;
      except
        CSL('UncQuant free error
i,j,name:'+i.ToString+', '+j.toString+', '+self.Name+ '
'+TMetricStore(fMetricList.Metrics.Items[i]).MetricDef.Name );
      end;
    end;

  end;
  inherited;
end;

procedure TCategoryMetrics.GenerateFinalOutput;
var i,j,k : integer;
    TMD : PMetricDef;
    TMO : TMetricStore;
    TSL : TStringList;
begin
  for i:=low(Results) to high(Results) do begin
    TMO:=TMetricStore(fMetricList.Metrics.Items[i]);
    TMD:=TMO.MetricDef;
    Results[i].Mean:=MeanUncertainty[i].Mean;
    Results[i].SumMean:=SumUncertaintyQuants[i].Mean;
    for j:=low(fConfig.UncQuantilePoints) to high(fConfig.UncQuantilePoints) do

```

```

begin
Results[i].UncMean[j]:=MeanUncertainty[i].Quantile(fConfig.UncQuantilePoints[j]);
Results[i].UncSum[j]:=SumUncertaintyQuants[i].Quantile(fConfig.UncQuantilePoints[j])
;
    end;
    for j:=low(fConfig.QuantilePoints) to high(fConfig.QuantilePoints) do begin
        Results[i].QuantileMeans[j]:=UncertaintyQuants[i,j].Mean;
        for k:=low(fConfig.UncQuantilePoints) to high(fConfig.UncQuantilePoints) do
begin
Results[i].UncQuantiles[j,k]:=UncertaintyQuants[i,j].Quantile(fConfig.UncQuantilePoi
nts[k]);
        end;
        if TMD.SaveByYear then begin
            for k:=0 to UncTmpYearly[i].Length-1 do
                Results[i].Yearly[k]:=UncTmpYearly[i].Values[k];
            end;
        end;
    end;
end;

procedure TCategoryMetrics.LoadFromStream(Strm: TStream);
var c,d,i,j : integer;
begin
    Strm.Read(i,sizeof(i));
    for c:=0 to i do begin
        if UncTmpyearly[c]<>nil then
            UncTmpyearly[c].LoadFromStream(Strm);
        end;

        Strm.Read(i,sizeof(i));
        for c:=0 to i do begin
            Strm.Read(j,sizeof(j));
            for d:=0 to j do begin
                UncertaintyQuants[c,d].LoadFromStream(Strm);
            end;
        end;

        Strm.Read(i,sizeof(i));
        for c:=0 to i do begin
            SumUncertaintyQuants[c].LoadFromStream(Strm);
        end;

        Strm.Read(i,sizeof(i));
        for c:=0 to i do begin
            MeanUncertainty[c].LoadFromStream(Strm);

```

```

end;
end;

function TCategoryMetrics.MemSize: integer;
var i,j,total : integer;
begin
  total:=0;
  for i:= low(VariabilityQuants) to high(VariabilityQuants) do
    total:=total+TStreamQuantiles(VariabilityQuants[i]).MemSize;
  for i:= low(SumUncertaintyQuants) to high(SumUncertaintyQuants) do
    total:=total+TStreamQuantiles(SumUncertaintyQuants[i]).MemSize;
  for i:= low(MeanUncertainty) to high(MeanUncertainty) do
    total:=total+TStreamQuantiles(MeanUncertainty[i]).MemSize;
  for i:= low(UncertaintyQuants) to high(UncertaintyQuants) do
    for j:= low(UncertaintyQuants[i]) to high(UncertaintyQuants[i]) do
      total:=total+TStreamQuantiles(UncertaintyQuants[i,j]).MemSize;
  Result:=Total;
end;

function TCategoryMetrics.NANINFCount: integer;
var i,j,total : integer;
begin
  total:=0;
  for i:= low(VariabilityQuants) to high(VariabilityQuants) do
    total:=total+TStreamQuantiles(VariabilityQuants[i]).fNANInfCounter;
  for i:= low(SumUncertaintyQuants) to high(SumUncertaintyQuants) do
    total:=total+TStreamQuantiles(SumUncertaintyQuants[i]).fNANInfCounter;
  for i:= low(MeanUncertainty) to high(MeanUncertainty) do
    total:=total+TStreamQuantiles(MeanUncertainty[i]).fNANInfCounter;
  for i:= low(UncertaintyQuants) to high(UncertaintyQuants) do
    for j:= low(UncertaintyQuants[i]) to high(UncertaintyQuants[i]) do
      total:=total+TStreamQuantiles(UncertaintyQuants[i,j]).fNANInfCounter;
  Result:=Total;
end;

procedure TCategoryMetrics.SaveResultsToStream(Strm: TStream);
var i : integer;
begin
  for i:=low(Results) to high(Results) do begin
    Results[i].SaveToStream(Strm);
  end;
end;

procedure TCategoryMetrics.SaveToStream(Strm: TStream);
var c,d,i,j : integer;
begin
  i:=high(UncTmpYearly);
  Strm.Write(i,sizeof(i));

```

LCRMetricCollector.pas

```

for c:=0 to i do begin
  if UncTmpeyearly[c]<>nil then
    UncTmpeyearly[c].SaveToStream(Strm);
end;

i:=high(UncertaintyQuants);
Strm.Write(i,sizeof(i));
for c:=0 to i do begin
  j:=high(UncertaintyQuants[c]);
  Strm.Write(j,sizeof(j));
  for d:=0 to j do begin
    UncertaintyQuants[c,d].SaveToStream(Strm);
  end;
end;

i:=high(SumUncertaintyQuants);
Strm.Write(i,sizeof(i));
for c:=0 to i do begin
  SumUncertaintyQuants[c].SaveToStream(Strm);
end;

i:=high(MeanUncertainty);
Strm.Write(i,sizeof(i));
for c:=0 to i do begin
  MeanUncertainty[c].SaveToStream(Strm);
end;
end;

{ TMetricSaver }

constructor TMetricSaver.create(aConfig : TLCRConfig; aMetricDef: PMetricDef);
var i : integer;
begin
  Mean:=0;
  SumMean:=0;

  setlength(UncMean,high(aConfig.UncQuantilePoints)+1);
  setlength(UncSum,high(aConfig.UncQuantilePoints)+1);
  setlength(QuantileMeans,high(aConfig.QuantilePoints)+1);
  setlength(UncQuantiles,high(aConfig.QuantilePoints)+1);
  for i:=0 to high(UncQuantiles) do begin
    setlength(UncQuantiles[i],high(aConfig.UncQuantilePoints)+1);
  end;
  if aMetricDef.SaveByYear then
    setlength(Yearly,aConfig.YearsOfAnalysis)
  else
    setlength(Yearly,0);

```



```

//TODO init arrays to 0?
end;

constructor TMetricSaver.createFromStream(Strm: TStream);
begin
    inherited;
    Mean:=0;
    SumMean:=0;
    LoadFromStream(Strm);
end;

destructor TMetricSaver.Destroy;
begin
    inherited;
end;

procedure TMetricSaver.LoadFromStream(Strm: TStream);
var i,j,k,l : integer;
begin
    Strm.Read(Mean,sizeof(Mean));
    Strm.Read(i,sizeof(i));
    Setlength(UncMean,i+1);
    for j:=0 to i do begin
        Strm.Read(UncMean[j],sizeof(double));
    end;

    Strm.Read(SumMean,sizeof(SumMean));
    Strm.Read(i,sizeof(i));
    Setlength(UncSum,i+1);
    for j:=0 to i do begin
        Strm.Read(UncSum[j],sizeof(double));
    end;

    Strm.Read(i,sizeof(i));
    Setlength(QuantileMeans,i+1);
    for j:=0 to i do begin
        Strm.Read(QuantileMeans[j],sizeof(double));
    end;

    Strm.Read(i,sizeof(i));
    Setlength(UncQuantiles,i+1);
    for j:=0 to i do begin
        Strm.Read(k,sizeof(k));
        Setlength(UncQuantiles[j],k+1);
        for l:=0 to k do
            Strm.Read(UncQuantiles[j,l],sizeof(double));
        end;
    end;
end;

```

```

Strm.Read(i,sizeof(i));
if i>=0 then begin
  SetLength(Yearly,i+1);
  for j:=0 to i do
    Strm.Read(Yearly[j],sizeof(double));
  end else
    SetLength(Yearly,0);
end;

procedure TMetricSaver.SaveToStream(Strm: TStream);
var i,j,k,l : integer;
begin
  Strm.Write(Mean,sizeof(Mean));
  i:=high(UncMean);
  Strm.Write(i,sizeof(i));
  for j:=0 to i do begin
    Strm.Write(UncMean[j],sizeof(double));
  end;

  Strm.Write(SumMean,sizeof(SumMean));
  i:=high(UncSum);
  Strm.Write(i,sizeof(i));
  for j:=0 to i do begin
    Strm.Write(UncSum[j],sizeof(double));
  end;

  i:=high(QuantileMeans);
  Strm.Write(i,sizeof(i));
  for j:=0 to i do begin
    Strm.Write(QuantileMeans[j],sizeof(double));
  end;
  i:=high(UncQuantiles);
  Strm.Write(i,sizeof(i));
  for j:=0 to i do begin
    k:=high(UncQuantiles[j]);
    Strm.Write(k,sizeof(k));
    for l:=0 to k do
      Strm.Write(UncQuantiles[j,l],sizeof(double));
    end;

  i:=high(Yearly);
  Strm.Write(i,sizeof(i));
  for j:=0 to i do
    Strm.Write(Yearly[j],sizeof(double));
end;

```

```

{ TCategoryList }

procedure TCategoryList.AddCategoryList(ACat: TcategoryList);
var i : integer;
begin
  for i:=0 to Categories.Count-1 do

TCategoryMetrics(Categories.Items[i]).AddCategoryMetrics(TCategoryMetrics(ACat.Categ
ories.Items[i]));
end;

function TCategoryList.AggPoints: int64;
var i : integer;
begin
  Result:=0;
  for i:=0 to Categories.Count-1 do
    Result:=Result+TCategoryMetrics(Categories.Items[i]).AggPoints;
end;

procedure TCategoryList.CollectUncertainty;
var i : integer;
begin
  GenerateSpecialMetrics;
  for i:=0 to Categories.Count-1 do
    TCategoryMetrics(Categories.Items[i]).CollectUncertainty;
end;

procedure TCategoryList.CollectVariability(const Cat: integer; const Wgt : double);
begin
  TCategoryMetrics(Categories.Items[Cat]).CollectVariability(wgt);
end;

constructor TCategoryList.Create(aConfig: TLCRConfig;
  aMetricList: TMetricList; aFileName : string);
var i : integer;
begin
  inherited create;
  Categories:=TObjectList.Create(true);
  fMetricList:=aMetricList;
  fConfig:=aConfig;
  //create Category for all
  Categories.Add(TCategoryMetrics.create(aConfig,aMetricList,' 0) All',''));
  for i:=0 to aConfig.OCategoryDefs.Count-1 do

Categories.Add(TCategoryMetrics.create(aConfig,aMetricList,TPicked(aConfig.oCategory
Defs.Items[i]).Name,TPicked(aConfig.oCategoryDefs.Items[i]).GetFilter));
  fFilename:=aFilename;
end;

```

```

destructor TCategoryList.Destroy;
begin
  try
    Categories.Free;
  except
    csl('CategoryList free error:');
  end;
  inherited;
end;

procedure TCategoryList.GenerateFinalOutput;
var i : integer;
begin
  for i:=0 to Categories.Count-1 do
    TCategoryMetrics(Categories.Items[i]).GenerateFinalOutput;
  end;

  procedure TCategoryList.GenerateSpecialMetrics;
  var j,i,cMCL,cDR : integer;
      tbv,tcv,bv,cv,v : double;
      appendstr1,appendstr2 : string;
      T : TMetricStore;
      nbpi,bcpi,anpi : integer;
      M : TCategoryMetrics;
  begin
    for i:=0 to Categories.Count-1 do begin
      M:=TCategoryMetrics(Categories.Items[i]);
      appendstr1:='
('+fConfig.OptionName+', '+floattostrf(fConfig.DiscountRate,ffffixed,10,2)+' )';
      appendstr2:=' ('+fConfig.OptionName+',0.00)';
      bv:=0; cv:=0;
      for j:=0 to fMetricList.Metrics.Count-1 do begin
        T:=TMetricStore(fMetricList.Metrics[j]);
        if CompareText(T.MetricDef.Name,'_Total Rule Cost'+Appendstr1)=0 then
          cv:=TCategoryMetrics(Categories.Items[i]).VariabilityQuants[j].Sum
        else
          if CompareText(T.MetricDef.Name,'Total Annual Benefits'+Appendstr1)=0
then
            bv:=TCategoryMetrics(Categories.Items[i]).VariabilityQuants[j].Sum
          else
            if CompareText(T.MetricDef.Name,'Benefit Cost Ratio'+Appendstr2)=0 then
              bcpi:=j
            else
              if CompareText(T.MetricDef.Name,'Annual Net Benefits'+Appendstr2)=0 then
                anpi:=j;
            end;
          end;
        end;
      end;
    end;
  end;

```

LCRMetricCollector.pas

```
    try
        v:=bv/cv;
    except
        v:=0;
    end;

    M.MeanUncertainty[bcpi].AddPoint(v);
    v:=bv-cv;
    M.MeanUncertainty[anpi].AddPoint(v);

end;
end;

procedure TCategoryList.LoadFromStream(AStream: TStream);
var i : integer;
begin
    for i:=0 to Categories.Count-1 do
        TCategoryMetrics(Categories.Items[i]).LoadFromStream(AStream);
    end;

function TCategoryList.MemSize: integer;
var i : integer;
begin
    Result:=0;
    for i:=0 to Categories.Count-1 do
        Result:=Result+TCategoryMetrics(Categories.Items[i]).MemSize;
    end;

function TCategoryList.NANINFCount: integer;
var i : integer;
begin
    Result:=0;
    try
        for i:=0 to Categories.Count-1 do
            Result:=Result+TCategoryMetrics(Categories.Items[i]).NANINFCount;
        except
            //error happeing atr end of run?
        end;
    end;

end;

procedure TCategoryList.SaveCategoryOutput;
var i,j : integer;
    Strm : TFileStream;
begin
    Strm:=TFileStream.Create(fFilename, fmCreate);
```

```

                                LCRMetricCollector.pas

//save metric defs;
i:=fMetricList.Metrics.Count;
Strm.Write(i,SizeOf(i));
for i:=0 to fMetricList.Metrics.Count-1 do

Strm.Write(PMetricDef(TmetricStore(fMetricList.Metrics.Items[i]).MetricDef)^,SizeOf(
TMetricDef));

//Save QuantilePoints
i:=high(fConfig.QuantilePoints);
Strm.Write(i,SizeOf(i));
for j:=0 to i do
    Strm.Write(fConfig.QuantilePoints[j],SizeOf(fConfig.QuantilePoints[j]));

i:=high(fConfig.UncQuantilePoints);
Strm.Write(i,SizeOf(i));
for j:=0 to i do
    Strm.Write(fConfig.UncQuantilePoints[j],SizeOf(fConfig.UncQuantilePoints[j]));

//Save category results
i:=Categories.Count;
Strm.Write(i,SizeOf(i));
for i:=0 to Categories.Count-1 do begin
    WriteStreamStr(Strm,TCategoryMetrics(Categories.Items[i]).Name);
    WriteStreamStr(Strm,TCategoryMetrics(Categories.Items[i]).Filter);
    TCategoryMetrics(Categories.Items[i]).SaveResultsToStream(Strm);
end;
Strm.Free;
end;

procedure TCategoryList.SaveToStream(AStream: TStream);
var i : integer;
begin
    for i:=0 to Categories.Count-1 do
        TCategoryMetrics(Categories.Items[i]).SaveToStream(AStream);
    end;

{ TCategoryOutputReader }

constructor TCategoryOutputReader.create(aFileName: string);
var Strm : TFileStream;
    j,i,k : integer;
    S : string;
    T : TMetricSaver;
begin
    inherited create;

```

LCRMetricCollector.pas

```

Names:=TStringList.Create;
Filters:=TStringList.Create;
Strm:=TFileStream.Create(aFilename, fmOpenRead);

//Read metric defs;
Strm.Read(i, SizeOf(i));
SetLength(Metrics, i);
for j:=0 to i-1 do
    Strm.Read(Metrics[j], SizeOf(TMetricDef));

//Read QuantilePoints
Strm.Read(i, SizeOf(i));
SetLength(QuantilePoints, i+1);
for j:=0 to i do
    Strm.Read(QuantilePoints[j], SizeOf(QuantilePoints[j]));

Strm.Read(i, SizeOf(i));
SetLength(UncQuantilePoints, i+1);
for j:=0 to i do
    Strm.Read(UncQuantilePoints[j], SizeOf(UncQuantilePoints[j]));
//Read category results
Strm.Read(i, SizeOf(i));
SetLength(Results, i);
for j:=0 to i-1 do begin
    Results[j]:=TObjectList.Create(true);
    S:=ReadStreamStr(Strm);
    Names.Add(S);
    S:=ReadStreamStr(Strm);
    Filters.Add(S);
    for k:=low(Metrics) to high(Metrics) do begin
        T:=TMetricSaver.createFromStream(Strm);
        Results[j].Add(T);
    end;
end;
Strm.Free;

end;

destructor TCategoryOutputReader.Destroy;
var i : integer;
begin
    for i:=low(Results) to high(Results) do
        Results[i].Destroy;
    Names.Free;
    Filters.Free;
end;

function TCategoryOutputReader.StripIt(S : string) : string;

```

```

var i : integer;
begin
  i:=Pos('(',S);
  result:=copy(S,1,i-3);
end;

function Strip5(s : string) : string;
var tmp : string;
begin
  tmp:=s;
  //  if tmp[4]=')' then
  //    system.delete(tmp,1,5);
  Result:=tmp;
end;

//TODO should be made a method....
function NameIt(q : byte) : string;
begin
  case q of
    mtCost:result:='Costs';
    mtBenefitDollars:result:='Benefit Dollars';
    mtBenefitCases:result:='Benefit Cases';
    mtNetBenefits:Result:='Net Benefits';
    mtBenefitCounts:Result:='Benefit Counts';
    mtInfo:Result:='Information';
    mtUMRA:Result:='UMRA Costs';
  else
    Result:='undefined';
  end;
end;

procedure TCategoryOutputReader.DumpMeanSumToDataset(DS: TDataset;
  TargMetricName: string; DoMean : boolean);
var i,j : integer;
    CatName,MetricName : string;
    T : TMetricSaver;
begin
  for i:=low(Results) to high(Results) do begin
    CatName:=Names[i];
    for j:=0 to Results[i].Count-1 do begin
      T:=TMetricSaver(Results[i].Items[j]);
      MetricName:=StripIt(Metrics[j].Name);
      if not (CompareText(MetricName,TargMetricName)=0) then continue;
    //do mean
      DS.Append;
      DS.FieldName('Category').AsString:=Strip5(CatName);
    end;
  end;
end;

```



```

                                LCRMetricCollector.pas
DS.FieldName('Option').AsString:=Metrics[j].Option;
DS.FieldName('DiscountRate').AsFloat:=Metrics[j].DiscRate;
if DoMean then begin
    DS.FieldName('P5').AsFloat:=T.UncMean[1];
    DS.FieldName('Mean').AsFloat:=T.Mean;
    DS.FieldName('P95').AsFloat:=T.UncMean[2];
end else begin
    DS.FieldName('P5').AsFloat:=T.UncSum[1];
    DS.FieldName('Mean').AsFloat:=T.SumMean;
    DS.FieldName('P95').AsFloat:=T.UncSum[2];
end;
DS.Post;
end;
end;
end;

procedure TCategoryOutputReader.DumpToDataset(DS: TDataset; OptionName: string);
//Note : Dataset must be opened and have proper field names defined
//Should also add proc to create an SQL string to create and populate
var i,j,ii : integer;
    CatName,MetricName : string;
    T : TMetricSaver;

begin
    for i:=low(Results) to high(Results) do begin
        CatName:=Names[i];
        for j:=0 to Results[i].Count-1 do begin
            T:=TMetricSaver(Results[i].Items[j]);
            MetricName:=StripIt(Metrics[j].Name);

            //Do quantile points
            for ii:=low(QuantilePoints) to high(QuantilePoints) do begin
                DS.Append;
                DS.FieldName('Category').AsString:=CatName;
                DS.FieldName('Metric').AsString:=MetricName;
                DS.FieldName('MetricClass').AsString:=NameIt(Metrics[j].MetricType);
                DS.FieldName('Option').AsString := OptionName;
                DS.FieldName('DiscountRate').AsFloat:=Metrics[j].DiscRate;
                DS.FieldName('Statistic').AsString:='P'+FloatToStr(QuantilePoints[ii]);

                DS.FieldName('P1').AsFloat:=T.UncQuantiles[ii,0];
                DS.FieldName('P5').AsFloat:=T.UncQuantiles[ii,1];
                DS.FieldName('Mean').AsFloat:=T.QuantileMeans[ii];
                DS.FieldName('P95').AsFloat:=T.UncQuantiles[ii,2];
                DS.FieldName('P99').AsFloat:=T.UncQuantiles[ii,3];
                DS.Post;
            end;
            //do mean and sum

```

```

DS.Append;
DS.FieldByName('Category').AsString:=CatName;
DS.FieldByName('Metric').AsString:=MetricName;
DS.FieldByName('MetricClass').AsString:=NameIt(Metrics[j].MetricType);
DS.FieldByName('Option').AsString := OptionName;
DS.FieldByName('DiscountRate').AsFloat:=Metrics[j].DiscRate;
DS.FieldByName('Statistic').AsString:='Mean';

```

```

DS.FieldByName('P1').AsFloat:=T.UncMean[0];
DS.FieldByName('P5').AsFloat:=T.UncMean[1];
DS.FieldByName('Mean').AsFloat:=T.Mean;
DS.FieldByName('P95').AsFloat:=T.UncMean[2];
DS.FieldByName('P99').AsFloat:=T.UncMean[3];
DS.Post;

```

```

DS.Append;
DS.FieldByName('Category').AsString:=CatName;
DS.FieldByName('Metric').AsString:=MetricName;
DS.FieldByName('MetricClass').AsString:=NameIt(Metrics[j].MetricType);
DS.FieldByName('Option').AsString := OptionName;
DS.FieldByName('DiscountRate').AsFloat:=Metrics[j].DiscRate;
DS.FieldByName('Statistic').AsString:='Sum';

```

```

DS.FieldByName('P1').AsFloat:=T.UncSum[0];
DS.FieldByName('P5').AsFloat:=T.UncSum[1];
DS.FieldByName('Mean').AsFloat:=T.SumMean;
DS.FieldByName('P95').AsFloat:=T.UncSum[2];
DS.FieldByName('P99').AsFloat:=T.UncSum[3];
DS.Post;

```

```

    end;
  end;
end;

```

```

procedure TCategoryOutputReader.Dump3VarToDataset(DS: TDataset;
  TargMetricName: string; s1, s2, s3: integer);
var i,j,ii : integer;
    CatName,MetricName : string;
    T : TMetricSaver;
begin
  for i:=low(Results) to high(Results) do begin
    CatName:=Names[i];
    for j:=0 to Results[i].Count-1 do begin
      T:=TMetricSaver(Results[i].Items[j]);
      MetricName:=StripIt(Metrics[j].Name);
      if not (CompareText(MetricName,TargMetricName)=0) then continue;

```

LCRMetricCollector.pas

```

//do mean
DS.Append;
DS.FieldByName('Category').AsString:=Strip5(CatName);
DS.FieldByName('Option').AsString:=Metrics[j].Option;
try
    DS.FieldByName('DiscountRate').AsFloat:=Metrics[j].DiscRate;
except
    //some reports do not have discount rate defined
end;

if S1=-1 then begin
    DS.FieldByName('S1P5').AsFloat:=T.UncMean[1];
    DS.FieldByName('S1Mean').AsFloat:=T.Mean;
    DS.FieldByName('S1P95').AsFloat:=T.UncMean[2];
end else begin
    DS.FieldByName('S1P5').AsFloat:=T.UncQuantiles[S1,1];
    DS.FieldByName('S1Mean').AsFloat:=T.QuantileMeans[S1];
    DS.FieldByName('S1P95').AsFloat:=T.UncQuantiles[S1,2];
end;

if S2=-1 then begin
    DS.FieldByName('S2P5').AsFloat:=T.UncMean[1];
    DS.FieldByName('S2Mean').AsFloat:=T.Mean;
    DS.FieldByName('S2P95').AsFloat:=T.UncMean[2];
end else begin
    DS.FieldByName('S2P5').AsFloat:=T.UncQuantiles[S2,1];
    DS.FieldByName('S2Mean').AsFloat:=T.QuantileMeans[S2];
    DS.FieldByName('S2P95').AsFloat:=T.UncQuantiles[S2,2];
end;

if S3=-1 then begin
    DS.FieldByName('S3P5').AsFloat:=T.UncMean[1];
    DS.FieldByName('S3Mean').AsFloat:=T.Mean;
    DS.FieldByName('S3P95').AsFloat:=T.UncMean[2];
end else begin
    DS.FieldByName('S3P5').AsFloat:=T.UncQuantiles[S3,1];
    DS.FieldByName('S3Mean').AsFloat:=T.QuantileMeans[S3];
    DS.FieldByName('S3P95').AsFloat:=T.UncQuantiles[S3,2];
end;

    DS.Post;
end;
end;
end;

procedure TCategoryOutputReader.DumpToTabDelim(FN: string; Mean: boolean;
NoICRCosts: boolean);
var i,j,ii : integer;

```

```

                                LCRMetricCollector.pas
CatName, MetricName, tmp : string;
T : TMetricSaver;
ttFile : TStreamWriter;
TheFile, TheFileU, TheFileI : TStreamWriter;
Tab : char;

begin
  Tab:=#9;
  tmp:=ChangeFileExt(FN, '');

  theFile:=TStreamWriter.Create(FN, false, TEncoding.ASCII, 4096);
  theFile.AutoFlush:=True;
  theFileI:=TStreamWriter.Create(tmp+'_icr.tab', false, TEncoding.ASCII, 4096);
  theFileI.AutoFlush:=True;
  theFileU:=TStreamWriter.Create(tmp+'_umra.tab', false, TEncoding.ASCII, 4096);
  theFileU.AutoFlush:=True;

  for i:=low(Results) to high(Results) do begin
    CatName:=Names[i];

    if i=low(Results) then begin
      //add column labels
      if Mean then begin
        TheFile.WriteLine('Category'+tab+'Metric'+tab+'Metric
Class'+tab+'Option'+tab+'Discount Rate'+tab+'Statistic'+tab+'Mean');
        TheFileU.WriteLine('Category'+tab+'Metric'+tab+'Metric
Class'+tab+'Option'+tab+'Discount Rate'+tab+'Statistic'+tab+'Mean');
        TheFileI.WriteLine('Category'+tab+'Metric'+tab+'Metric
Class'+tab+'Option'+tab+'Discount Rate'+tab+'Statistic'+tab+'Mean');
      end else begin
        TheFile.WriteLine('Category'+tab+'Metric'+tab+'Metric
Class'+tab+'Option'+tab+'Discount
Rate'+tab+'Statistic'+tab+'P1'+tab+'P5'+tab+'Mean'+tab+'P95'+tab+'P99');
        TheFileI.WriteLine('Category'+tab+'Metric'+tab+'Metric
Class'+tab+'Option'+tab+'Discount
Rate'+tab+'Statistic'+tab+'P1'+tab+'P5'+tab+'Mean'+tab+'P95'+tab+'P99');
        TheFileU.WriteLine('Category'+tab+'Metric'+tab+'Metric
Class'+tab+'Option'+tab+'Discount
Rate'+tab+'Statistic'+tab+'P1'+tab+'P5'+tab+'Mean'+tab+'P95'+tab+'P99');
      end;
    end;

    for j:=0 to Results[i].Count-1 do begin
      T:=TMetricSaver(Results[i].Items[j]);
      MetricName:=StripIt(String(Metrics[j].Name));

      if (MetricName='System Find and Fix Total_P') or
        (MetricName='System Find and Fix Total_PWS') or

```

```

LCRMetricCollector.pas
(MetricName='System Lead Install CCT Total_P') or
(MetricName='System Lead Install CCT Total_PWS') or
(MetricName='System Lead Modify CCT Total_P') or
(MetricName='System Lead Modify CCT Total_PWS')
then continue;

if AnsiPos('ICR', MetricName) > 0 then ttFile:=TheFileI else
if AnsiPos('UMRA', MetricName) > 0 then ttFile:=TheFileU else
ttFile:=theFile;

//Do quantile points
for ii:=low(QuantilePoints) to high(QuantilePoints) do begin
if Mean then

ttFile.WriteLine(CatName+tab+MetricName+tab+NameIt(Metrics[j].MetricType)+tab+Metrics[j].Option+tab+FloatToStrF(Metrics[j].DiscRate,fffFixed,3,2)+tab+'P'+FloatToStr(trunc(QuantilePoints[ii]*100))+tab+FloatToStrF(T.QuantileMeans[ii],ffExponent,10,2))
else

ttFile.WriteLine(CatName+tab+MetricName+tab+NameIt(Metrics[j].MetricType)+tab+Metrics[j].Option+tab+FloatToStrF(Metrics[j].DiscRate,fffFixed,3,2)+tab+'P'+FloatToStr(trunc(QuantilePoints[ii]*100))+tab+FloatToStrF(T.UncQuantiles[ii,0],ffExponent,10,2)+tab+FloatToStrF(T.UncQuantiles[ii,1],ffExponent,10,2)+tab+FloatToStrF(T.QuantileMeans[ii],ffExponent,10,2)+tab+FloatToStrF(T.UncQuantiles[ii,2],ffExponent,10,2)+tab+FloatToStrF(T.UncQuantiles[ii,3],ffExponent,10,2));
end;

//do mean and sum
if Mean then

ttFile.WriteLine(CatName+tab+MetricName+tab+NameIt(Metrics[j].MetricType)+tab+Metrics[j].Option+tab+FloatToStrF(Metrics[j].DiscRate,fffFixed,3,2)+tab+'Mean'+tab+FloatToStrF(T.Mean,ffExponent,10,2))
else

ttFile.WriteLine(CatName+tab+MetricName+tab+NameIt(Metrics[j].MetricType)+tab+Metrics[j].Option+tab+FloatToStrF(Metrics[j].DiscRate,fffFixed,3,2)+tab+'Mean'+tab+FloatToStrF(T.UncMean[0],ffExponent,10,2)+tab+FloatToStrF(T.UncMean[1],ffExponent,10,2)+tab+FloatToStrF(T.Mean,ffExponent,10,2)+tab+FloatToStrF(T.UncMean[2],ffExponent,10,2)+tab+FloatToStrF(T.UncMean[3],ffExponent,10,2));

if Mean then

ttFile.WriteLine(CatName+tab+MetricName+tab+NameIt(Metrics[j].MetricType)+tab+Metrics[j].Option+tab+FloatToStrF(Metrics[j].DiscRate,fffFixed,3,2)+tab+'Sum'+tab+FloatToStrF(T.SumMean,ffExponent,10,2))
else

```

LCRMetricCollector.pas

```
ttFile.WriteLine(CatName+tab+MetricName+tab+NameIt(Metrics[j].MetricType)+tab+Metrics[j].Option+tab+FloatToStrF(Metrics[j].DiscRate,ffFixed,3,2)+tab+'Sum'+tab+FloatToStrF(T.UncSum[0],ffExponent,10,2)+tab+FloatToStrF(T.UncSum[1],ffExponent,10,2)+tab+FloatToStrF(T.SumMean,ffExponent,10,2)+tab+FloatToStrF(T.UncSum[2],ffExponent,10,2)+tab+FloatToStrF(T.UncSum[3],ffExponent,10,2));
    end;
end;

TheFile.Free;
TheFileU.Free;
TheFileI.Free;
end;

end.
```