```
unit LCRResultsFile;

interface

uses SysUtils, Classes, DB, LCRGlobals, MtxVec,Math387,
dialogs,
     Agglomerators;

//this is for MtxVec stuff
{$R-}


const
 PercentileValuesToStore =20;

type

  TMetric=record
    ContamLevel : integer;
    Value,Probability : double;
  end;

  TtmpMetrics=array of TMetric;
  PtmpMetrics=^TtmpMetrics;

  TMetricDef=record
    Name : string[100];
    Option: string[100];
    DiscRate : single;
    SaveVariability,SaveByYear,IgnoreZeros,SingleValue,
    CategoryOnly : boolean;
    MetricType    : byte;
  end;
  PMetricDef=^TMetricDef;

  TMetricOutput=class
  private
    fSaveYear : boolean;
    fMetricDef : PMetricDef;
    PercInc,PercStart  : double;
    tmpMetrics : TtmpMetrics;
  public
    OutputSize : integer;
    Results : TMtx;

    CurMean : double;
    CurPercs : TVec;
    CurYearly : TVec;
```

```
    constructor Create(const Def : TMetricDef; Years,Levels : integer);
    destructor Destroy; override;

    procedure Reset;
    procedure CreateTmpMetricList(var N : integer);

    procedure LoadFromStream(Strm : TStream);
    procedure SaveToStream(Strm : TStream);

    procedure ApplyDist(const Conc: TVec; const Threshold : double);
  end;


  TLCRResultsFile=class
  private
    fNumMetrics : integer;
    fNumindexRecs : integer;
    fYears,fLevels  : integer;
  public

    Metrics : TStringList;
    MetricResults : TStringList;
    CurrentID      : string;

    constructor Create(const FileName: string; Years,Levels :integer);
    destructor Destroy; override;


    function AddMetricDef(Name: string; MetricType: byte;
SaveVar,SaveYear,IgnoreZeros: boolean;
                          aOption: string; ADiscRate: single;
SaveSingleValue,UncOnly: boolean) : PMetricDef;

    //debug procedure
    procedure ReadRawToDataset(Sample,Limit : integer; DS : TDataset);

    procedure ResetOutputs;

    procedure DumpContents(S : string);
  end;


procedure BSortTmpMetrics(tmpResults : PtmpMetrics; xLow,xHi : Integer);
procedure QSortTmpMetrics(var tmpResults : TTmpMetrics; iLo, iHi: Integer);

procedure GenVariabilityPercs(const tmpResults : TtmpMetrics;
                              var tmpPercVec : TVec;
```

```
                              var TotalProbSum : double; Start, Stop : integer);


implementation

procedure BSortTmpMetrics(tmpResults : PtmpMetrics; xLow,xHi : Integer);
var
  I, J: Integer;
  dummy : TMetric;
begin

  for I := xHi downto xLow do
    for J := xLow to xHi - 1 do
      if tmpResults^[J].Value > tmpResults^[J + 1].Value then begin
        Dummy := tmpResults^[j];
        tmpResults^[j] := tmpResults^[j+1];
        tmpResults^[j+1] := Dummy;
      end;
end;

{$IFOPT R-}
  {$DEFINE RANGEOFF}
  {$R+}
{$ELSE}
  {$UNDEF RANGEOFF}
{$ENDIF}



procedure QSortTmpMetrics(var tmpResults : TtmpMetrics; iLo, iHi: Integer);
var Lo, Hi : Integer;
    Val: Double;
    Dummy: TMetric;
begin
  Lo := iLo;
  Hi := iHi;

  Val := tmpResults[(Lo + Hi) div 2].Value;
  repeat
    while ( tmpResults[lo].Value < Val ) do begin
      Inc(lo);
    end;
    while ( tmpResults[hi].Value > Val ) do begin
      Dec(hi);
    end;
    if ( lo <= hi ) then begin
      Dummy := tmpResults[lo];
      tmpResults[lo] := tmpResults[hi];
```

```
      tmpResults[hi] := Dummy;
      Inc(lo);
      Dec(hi);
    end;
  until ( lo > hi );

  if Hi > iLo then QSortTmpMetrics(TmpResults, iLo, Hi);
  if Lo < iHi then QSortTmpMetrics(TmpResults, Lo, iHi);
end;


procedure GenVariabilityPercs(const tmpResults : TtmpMetrics;
                              var tmpPercVec : TVec;
                              var TotalProbSum : double; Start, Stop : integer);
var
  n,i: integer;
  ProbSum,Perc, w1, w2, d1, d2: double;
  fPercInc:double;
begin
  tmpPercVec.SetZero;
  TotalProbSum:=0;
  if Stop<Start then exit;

  //Normalize probs...
  ProbSum:=0;
  for i:=Start to Stop do
    ProbSum:=ProbSum+tmpResults[i].Probability;
  TotalProbSum:=ProbSum;
  if TotalProbSum=0 then begin
     //set all to zero and exit...
     for i:=0 to PercentileValuesToStore-1 do begin
       tmpPercVec.Values[i]:= 0;
     end;
     exit;
  end;


  fPercInc:=1/PercentileValuesToStore;

  for i:=Start to Stop do
    tmpResults[i].Probability:=tmpResults[i].Probability/ProbSum;

  //Find percentiles
  n:=Start;
  ProbSum:=0;
  for i:=0 to PercentileValuesToStore-1 do begin
    Perc:=fPercInc/2+(i)*fPercInc;
    while ( (n <= Stop) and (ProbSum < Perc) ) do begin
```

```
      ProbSum := ProbSum + tmpResults[n].Probability;
      inc(n);
    end;
    if ((ProbSum = Perc) or (n=1)) then
      tmpPercVec.Values[i]:= tmpResults[n-1].Value
    else begin
      //apply inverse-distance weighted interpolation
      d1:=(Perc - (ProbSum - tmpResults[n-1].Probability));
      d2:=(ProbSum - Perc);
      if d1=0 then begin
         w1:=1; w2:=0;
      end else
      if d2=0 then begin
         w1:=0; w2:=1;
      end else begin
        w1 := 1/d1;
        w2 := 1/d2;
      end;
      tmpPercVec.Values[i] :=  ((tmpResults[n-2].Value * w1) +
(tmpResults[n-1].Value * w2))/(w1+w2);

    end;
  end;
end;

{$IFDEF RANGEOFF}
  {$R-}
  {$UNDEF RANGEOFF}
{$ENDIF}




{ TLCRResultsFile }

function TLCRResultsFile.AddMetricDef(Name: string; MetricType: byte;
SaveVar,SaveYear,IgnoreZeros: boolean;
                                      aOption: string; aDiscRate: single;
SaveSingleValue,UncOnly: boolean): PMetricDef;
var PMDef : PMetricDef;
    MetricObj : TMetricOutput;
    appendstr : string;

begin
  New(PMDef);
  Result:=PMDef;

  appendstr:='  ('+aOption+','+floattostrf(aDiscRate,fffixed,10,2)+')';
```

```
  PMDef^.Name:=Name+appendstr;

  PMDef^.SingleValue:=SaveSingleValue;
  PMDef^.SaveVariability:=SaveVar;
  PMDef^.SaveByYear:=SaveYear;
  PMDef^.CategoryOnly:=UncOnly;
  PMDef^.IgnoreZeros:=IgnoreZeros;
  PMDef^.MetricType:=MetricType;
  PMDef^.Option:=aOption;
  PMDef^.DiscRate:=aDiscRate;
  Metrics.AddObject(PMDef^.Name,Pointer(PMDef));

  //Add output object
  MetricObj:=TMetricOutput.Create(PMDef^,fYears, fLevels);
  MetricResults.AddObject(PMDef^.Name,MetricObj)
end;

constructor TLCRResultsFile.Create(const FileName: string; Years,Levels : integer);
var T : file;
    i : integer;
begin
  inherited Create;

  fYears:=Years;
  fLevels:=Levels;

  Metrics:=TStringList.Create;
  Metrics.Sorted:=True;
  //Metrics.Duplicates:=dupAccept;
  Metrics.Duplicates:=dupIgnore;

  MetricResults:=TStringList.Create;
  MetricResults.Sorted:=True;
  //MetricResults.Duplicates:=dupAccept;
  MetricResults.Duplicates:=dupIgnore;


end;


destructor TLCRResultsFile.Destroy;
var i,j : integer;
begin
  for i:=0 to Metrics.Count-1 do dispose(PMetricDef(Metrics.Objects[i]));
  for i:=0 to MetricResults.Count-1 do TMetricOutput(MetricResults.Objects[i]).Free;
  Metrics.Free;
  MetricResults.Free;
```

```
  inherited;
end;

procedure TLCRResultsFile.DumpContents(S: string);
var c,i,j : integer;
    T : TMetricOutput;
    SF : TextFile;
begin
   (*
   assignfile(SF,S);
   rewrite(SF);
   for c:=0 to high(fMaps) do begin
     for i:=0 to fMaps[c].RecCount-1 do begin
       fMaps[c].CurrentRec:=i;
       fMaps[c].ReadRec;
       for j:=0 to MetricResults.Count-1 do begin
        T:=TMetricOutput(MetricResults.Objects[j]);
        writeln(SF,T.fMetricDef^.Name);
       end;
     end;
   end;
   closefile(SF);
   *)
end;

procedure TLCRResultsFile.ResetOutputs;
var i : integer;
begin
  for i:=0 to MetricResults.Count-1 do
    TMetricOutput(MetricResults.Objects[i]).Reset;
end;


procedure TLCRResultsFile.ReadRawToDataset(Sample,Limit : integer; DS : TDataset);
var
    tmpVec : TVec;
    ID : string;
    cnt,i,ii,j : integer;
begin
  (*
  createit(tmpVec);
  cnt:=0;

  for ii:=0 to IndexList.Count-1 do begin
    cnt:=PIdxRec(IndexList.Objects[ii])^.RecNo;
    CurrentRecord(cnt);
    if ii>limit then break;
```

```
    ReadRec;
    j:=0;
    while j<TMetricOutput(MetricResults.Objects[0]).Results.Cols-1 do begin

      for i:=0 to MetricResults.Count-1 do begin

        DS.Append;
        DS.FieldByName('ID').AsString:=IndexList[ii];
        DS.FieldByName('Conc').AsFloat:=j;
        DS.FieldByName('Name').AsString:=MetricResults.Strings[i];
        DS.FieldByName('Sample').AsInteger:=sample;

DS.FieldByName('DiscRate').AsFloat:=TMetricOutput(MetricResults.Objects[i]).fMetricD
ef.DiscRate;

DS.FieldByName('MCL').AsFloat:=TMetricOutput(MetricResults.Objects[i]).fMetricDef.MC
L;

        if PMetricDef(Metrics.Objects[i])^.SaveVariability then begin

DS.FieldByName('P0').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[0,j];

DS.FieldByName('P5').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[1,j];

DS.FieldByName('P50').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[10,j]
;

DS.FieldByName('P95').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[19,j]
;
        end else
        if PMetricDef(Metrics.Objects[i])^.SaveByYear then begin

DS.FieldByName('Y20').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[20,j]
;

DS.FieldByName('Y0').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[0,j];

DS.FieldByName('Y5').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[5,j];

DS.FieldByName('YLast').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[TMe
tricOutput(MetricResults.Objects[i]).Results.Rows-1,j];
        end else begin

DS.FieldByName('Mean').AsFloat:=TMetricOutput(MetricResults.Objects[i]).Results[0,j]
;
        end;

        DS.Post;
```

```
      end;
      j:=j+5;
    end;
  end;
  freeit(tmpvec);
  *)
end;


{ TMetricOutput }

procedure TMetricOutput.ApplyDist(const Conc: TVec; const Threshold : double);
var i,j,k : integer;
    TotalProbSum : double;

{}dp:integer;

begin

try
  CurMean:=0;
  TotalProbSum:=0;
  if fMetricDef.SaveByYear then
    CurYearly.SetZero
  else
    CurPercs.SetZero;

{}dp:=0;
  for i:=0 to Conc.Length-1 do begin
    //stop calcs when prob is less than user specified threshold
    //Removed 12/17/13 because FL files can have discontinuities...
    (*
    if i>1 then begin
{}dp:=10;
     if (Conc.Values[i]<Conc.Values[i-1]) and
        (Conc.Values[i]<Threshold) then break;

    end;
    *)

    if fMetricDef.SaveByYear then begin
{}dp:=11;
      for j:=0 to CurYearly.Length-1 do begin
        CurYearly.Values[j]:=CurYearly.Values[j]+Results.Values[j,i]*Conc.Values[i];
      end;
{}dp:=12;
      CurMean:=CurMean+Results.Values[0,i]*Conc.Values[i];
    end else
```

```
    if fMetricDef.SaveVariability then begin
       //for j:=0 to Results.Rows-1 do
       //  PercAgglom.AddPoint(Results[j,i],Conc.Values[i]);
    end else
    begin
{}dp:=13;
      CurMean:=CurMean+Results.Values[0,i]*Conc.Values[i];
      if (fMetricDef.IgnoreZeros) and (Results.Values[0,i]>0) then
TotalProbSum:=TotalProbSum+Conc.Values[i];
    end;
  end;

  //this adjusts for "ignorezero" in regular matrics...
  if (TotalProbSum>0) and (CurMean>0) then CurMean:=CurMean/TotalProbSum;

{}dp:=14;
  //Set final "variability" percentiles
  if fMetricDef.SaveVariability then begin
    //TODO:  CreateTmpMetricList will be called multiple times in the "# Occ Pulls"
type runs.
    //That isn't necc., but we will probably depricate that method anyway.
{}dp:=141;
    CreateTmpMetricList(j);
    for i:=0 to j do begin
{}dp:=142;
      tmpMetrics[i].Probability:=Conc.Values[tmpMetrics[i].ContamLevel];
    end;
{}dp:=143;
    GenVariabilityPercs(tmpMetrics,CurPercs,TotalProbSum,0,j);
  end;
except
on e:exception do
  raise
exception.Create('Error:'+fMetricDef^.Name+':'+inttostr(dp)+',i:'+inttostr(i)+',j:'+
inttostr(j)+',rc:'+inttostr(Results.Cols)+',rr:'+inttostr(Results.rows)+'
'+e.message);
end;

end;

constructor TMetricOutput.Create(const Def : TMetricDef; Years,Levels : integer);
var T : TMemoryStream;
begin
  inherited create;
  OutputSize:=0;
  T:=TMemoryStream.Create;
  fMetricDef:=@Def;
  fSaveYear:=Def.SaveByYear;
```

```
  Results:=TMtx.Create;
  //TODO using this here will probably prove to be a bottleneck.
  //PercAgglom:=TStreamQuantiles.create(0.01);

  PercInc:=1/PercentileValuesToStore;
  PercStart:=PercInc/2;

  if Def.SaveByYear then begin
    Results.Size(Years,Levels);
    CurYearly:=TVec.Create;
    CurYearly.Size(Years);
  end else begin
    CurPercs:=TVec.Create;
    CurPercs.Size(PercentileValuesToStore);
    if Def.SaveVariability then begin
      Results.Size(PercentileValuesToStore,Levels);
      SetLength(tmpMetrics,PercentileValuesToStore*Levels);
    end else
      Results.Size(1,Levels);
  end;

  SaveToStream(T);
  OutputSize:=T.Size;
  T.Free;
end;

procedure TMetricOutput.CreateTmpMetricList(var N : integer);
var i,j,c,dp : integer;
begin
  N:=-1;
  if not fMetricDef.SaveVariability then exit;
  c:=0;

try
dp:=1;

  for i:=0 to Results.Cols-1 do begin
    for j:=0 to Results.Rows-1 do begin
      if ((abs(Results.Values[j,i])>1e-6) or (not fMetricDef.IgnoreZeros)) then
begin

dp:=2;

        tmpMetrics[c].ContamLevel:=i;
        tmpMetrics[c].Probability:=1;
        tmpMetrics[c].Value:=Results.Values[j,i];
        inc(c);
```

```
      end;
    end;
  end;

dp:=3;

  if C>1 then
    qSortTmpMetrics(tmpMetrics,0,c-1);

except
    on e:exception do begin
      raise exception.create('in createtmp dp:'+inttostr(dp)+' c:'+inttostr(c)+'
'+e.Message);
    end;
end;

  N:=c-1;
end;

destructor TMetricOutput.Destroy;
begin
  Results.Free;
  //PercAgglom.Free;
  if fSaveYear then
    CurYearly.Free
  else
    CurPercs.Free;
  inherited;
end;

procedure TMetricOutput.LoadFromStream(Strm: TStream);
begin
  Results.LoadFromStream(Strm);
end;

procedure TMetricOutput.Reset;
begin
  Results.SetZero;
  if fMetricDef.SaveByYear then
    CurYearly.SetZero
  else
    CurPercs.SetZero;
end;

procedure TMetricOutput.SaveToStream(Strm: TStream);

begin
  //TODO saving as single doubel should be made optional
```

```
  Results.SaveToStream(Strm)
end;

end.
```