```
unit LCRGlobals;

interface

uses SysUtils, StrUtils, Classes, Generics.Collections,
     //    {$IFDEF DEBUG}
      CodeSiteLogging,
      //    {$ENDIF}
     Math;

const
   dNone       = 299;
   dCustom     = 300;
   dNormal     = 301;   dTriangular  = 302;   dPoisson  = 303;
   dBinomial   = 304;   dLogNormal   = 305;   dUniform  = 306;
   dExponential= 307;   dGeometric   = 308;   dWeibull  = 309;
   dGamma      = 310;   dLogistic    = 311;   dCauchy   = 314;
   dPareto     = 313;   dBeta        = 312;   dHML      = 315;
   dStudentT   = 316;
   dVariableCustom  = 317; dICustom = 318;

   ADOConStr='Provider=Microsoft.ACE.OLEDB.12.0;Data Source=%s;Persist Security
Info=False;';


type

  TSingleArray = Array of Single;
  PSingleArray = ^TSingleArray;
  TDoubleArray = Array of Double;
  PDoubleArray = ^TDoubleArray;

  TDoubleArrayDouble = Array of TDoubleArray;
  TBigArrayDouble = Array of TDoubleArrayDouble;
  TBigDistributionArray = Array of Array of TBigArrayDouble;

  TYearlyMovement = array of array[1..16] of double;
  TYearlyMovementMicro = array of array[1..16,1..16] of double;
  TMovementMicro = array[1..16,1..16] of double;

  TRateArray = array[1..2,0..85] of double; //gender by age
  TSubpopBoolean = array[1..2,0..85] of boolean;//gender and age
  TSubPopArray = array[1..2,0..85] of single;//gender and age
  PSubPopArray = ^TSubPopArray;

  TYearlySubPopArray = array of TSubPopArray;
  PYearlySubPopArray = ^TYearlySubPopArray;
```

```
//--- TState Enumerated values match the State FIPS codes ---//
  TState = (sAlabama=1, sAlaska, sArizona=4, sArkansas, sCalifornia, sColorado=8,
sConnecticut,
          sDelaware, sDistrictOfColumbia, sFlorida, sGeorgia, sHawaii=15, sIdaho,
sIllinois,
          sIndiana, sIowa, sKansas, sKentucky, sLouisiana, sMaine, sMaryland,
sMassachusetts,
          sMichigan, sMinnesota, sMississippi, sMissouri, sMontana, sNebraska,
sNevada,
          sNewHampshire, sNewJersey, sNewMexico, sNewYork, sNorthCarolina,
sNorthDakota,
          sOhio, sOklahoma, sOregon, sPennsylvania, sRhodeIsland=44, sSouthCarolina,
          sSouthDakota, sTennessee, sTexas, sUtah, sVermont, sVirginia,
sWashington=53,
          sWestVirginia, sWisconsin, sWyoming, sAmericanSamoa, sGuam,
sNorthernMarianas,
          sPalau, sPuertoRico,sTribes,sVirginIslands, sUnknown);

//--- A TRegion is just a collection of one or more TStates ---//
  TRegion = record
     Region: string;
     RegionID: integer;
     States: Array of TState;
  end;

//--- A TRegionArray is just a collection of one or more TRegion ---//
  TRegionArray = array of TRegion;

//--- TRace, TGender, TAge - used to represent Population fields ---//
  TRace   = (rWhite,rBlack,rOther);
  TGender = (gMale,gFemale);
  TAge    = (a0to4,a5to9,a10to14,a15to19,a20to24,a25to29,a30to34,a35to39,
             a40to44,a45to49,a50to54,a55to59,a60to64,a65to69,a70to74,a75to79,
             a80to84,a85U);

//--- TPopulationData - holds individual population statistics and a total
population ---//
  TPopulationData = record
     Statistics: Array[gMale..gFemale,a0to4..a85U] of Double;
     TotalPopulation: Double;
  end;

//--- TSystemType - used to represent the type of water system (community,
non-transient, etc.) ---//
  TSystemType = (sysCWS, sysNTNC, sysTNC);
```

```
//--- TOwnership - used to represent the system ownership ---//
  TOwnership = (oPublic, oPrivate);

//--- TSourceWater - used to represent the source of the system's water ---//
  TSourceWater = (swGroundwater, swSurfacewater);

//--- TSystemSize - used to represent the size of the system in number of people
served ---//
  TSystemSize =
(ss0_100,ss101_500,ss501_1000,ss1001_3300,ss3301_10k,ss10k_50k,ss50k_100k,ss100k_1m,
ss1m);

//--- TContaminantType - used to represent the contaminant for monitoring---//
  TContaminantType = (cIOC, cSOC, cVOC, cNitrate, cNitrite, cRadionuclides);

//--- TPWSRecord - holds individual system information from the SDWIS data base
---//
  TPWSRecord = record
     PWSId : string[200];
     SystemSize : TSystemSize;
     SystemType : TSystemType;
     Ownership : TOwnership;
     SourceWater : TSourceWater;
     Region : Integer;
     Population : double;
  end;

  TPicked=class //support for category picker
    Name : string;
    Size,Region : array[1..11] of boolean;
    Owner,Source,SysType : array[1..2] of boolean;
    constructor create;
    function GetName : string;
    function GetFilter : string;
    procedure SaveToStream(Strm : TStream);
    procedure LoadFromStream(Strm : TStream);
  end;

  TPercentiles = array[0..99] of double;
  PPercentiles = ^TPercentiles;

  //Handles User inputed variability and uncertainty files
  TSafeWaterPercUncFile=class
    class function ValidFile(aFile : string) : boolean;
    class function GetRandom(aFile : string; var P : TPercentiles) : double;
    class function GetMean(aFile : string; var P : TPercentiles) : double;
    class function GetIX(aFile : string; aIX : integer; var P : TPercentiles) :
double;
```

```
  end;

  TSWVar=record
    Name : string[25];
    Description : string[200];
    V : double;
    P : TPercentiles;
    PFile : string[200];
    PointValue,Parm1,Parm2,Parm3,LowerBound,UpperBound,CensorValue : double;
    Distribution : integer;
  end;
  PSWVar = ^TSWVar;

  TStateDataRecord = record
    StateCode:  string[2];
    PWSCount: integer;
  end;
  TStateDataArray = array [0..69] of TStateDataRecord;
  PStateDataArray = ^TStateDataArray;

  // For reading Data Request database
  TDoubleArray2 = Array of TDoubleArray;
  TDoubleArray3 = Array of TDoubleArray2;
  TDoubleArray4 = Array of TDoubleArray3;

  // Data Need Type
  TDataNeedType = (dnCost, dnCostFunction, dnCount, dnDemographic, dnDistribution,
                   dnHours, dnRate, dnNumSamp, dnPercentage, dnProbability);

  // State / System Cost
  TCostBase = (cbEP, cbState, cbSystem);

  // Unit Basis for Cost
  TUnitBasis = (ubOneTime, ubPerYear, ubPerYearVariable, ubSystem, ubBlank);

  // Distribution Type
  TDistributionType = (dtPointEstimate, dtUniform, dtTriangular, dtNormal,
dtLogNormal,
                       dtFunction, dtCustom);

  TTreatmentFunctionalForm = (ffPow=1,ffLN,ffExp,ffPoly);

  TCostGenRec=record
    PWSid: string;
    SourceWater,SystemSize,Ownership,SystemType : integer;

    EntryPoints : integer;
    AvgRevenue, PWSAnnualRevenue : double;
```

```
    Population: integer;
    CostCapital   : double;
    InflatedPops : PSingleArray;
    SamplingWeight : double;

    LSL, CCT: integer;
    Connections: integer;
    NumProxies : integer;
    First_ale: integer;

    DFlowEP,AFlowEP : double;
    NumberLSLs: double;

    CCTPO4, CCTPH, CCTBoth,
    BaselinePO4Dose,Baselineph_wPh, Baselineph_woPh,
    Baselineph_woCCT, Baselineph_wPO4ph: integer;

    fBaseVars,fScenVars : TDictionary<string,double>;
    P90_base: double;

    class function HeaderString : string; static;
    function DataString : string;


  end;

  TAddCostGenRec=record
    Small_Correct, Num_Proxies: integer;
    Bin: integer;
  end;

  TVLSEpWorkbookRec = record
    p_b3: integer;
    baselinePH_woCCT, baselinePH_wPh, baselinePH_woPh, Baselineph_wpo4ph,
baselinePO4Dose: integer;
    pbasephpo4, pbaseph, pbasepo4: integer;
    Connections: integer;
    CCT, LSL: integer;
    NumberEPs, Population, NumberLSLs: integer;
    AFlowEp, DFlowEp: double;
  end;

  TBaselineSavedValues = record
    PWSId2: string;
    cost_pickup_samp: string;
    numb_ha: string;
    numb_lcr_other_org: string;
    pp_hh_return_samp: string;
```

```
    cost_commercial_lab: string;
  end;

var
  HelpPath,DataPath,UserPath,AppPath : string;
  WindowsVersion,ProcessorCount,LogicalProcessors  : integer;
  PreCalcDiscRate : array[0..100] of double;
  StateNames : TStringList;

  //silly tables to assist category picker....
  zSizeNames,zSizeFilters : array[1..9] of string;
  zRegNames,zRegFilters : array[1..10] of string;
  zOwnerNames,zOwnerFilters : array[1..2] of string;
  zSrcNames,zSrcFilters : array[1..2] of string;
  zTypeNames,zTypeFilters : array[1..2] of string;

  //global to see what bin movments are possible and where they wind up (CCT,LSL)

  GMoveBinLC : TMovementMicro;

const

StateCommaList='AK,AL,AR,AS,AZ,CA,CO,CT,DC,DE,FL,GA,GU,HI,IA,ID,IL,IN,KS,KY,LA,MA,MD
,ME,MI,MN,MO,MP,MS,MT,NC,ND,'+

'NE,NH,NJ,NM,NV,NY,OH,OK,OR,PA,PR,RI,SC,SD,TN,TX,UT,VA,VI,VT,WA,WI,WV,WY,' +
                 '01,02,03,04,05,06,07,08,09,10,NN';

function TSystemSizeToStr(size:TSystemSize): string;
function TSourceWaterToStr(source:TSourceWater): string;
function TOwnershipToStr(owner: TOwnership): string;
function TSystemTypeToStr(systype: TSystemType): string;
function StrToTSystemSize(value:String): TSystemSize;
function StrToTSourceWater(value:string): TSourceWater;
function StrToTOwnership(value: string): TOwnership;
function StrToTSystemType(value:string): TSystemType;
function IntToTAge(Age: Integer): TAge;
function TGenderToStr(Gender: TGender): string;
function TAgeToStr(Age: TAge): string;
function StrSAToTState(StateString: string): TState;

procedure WriteStreamStr(Stream : TStream; Str : string);
function ReadStreamStr(Stream : TStream) : string;

// For reading Data Request database
function TDataNeedTypeToStr(v: TDataNeedType): string;
function TDataNeedTypeFromStr(v: string): TDataNeedType;
function TCostBaseToStr(v: TCostBase): string;
```

```
function TCostBaseFromStr(v: string): TCostBase;
function TUnitBasisToStr(v: TUnitBasis): string;
function TUnitBasisFromStr(v: string): TUnitBasis;
function TDistributionTypeToStr(v: TDistributionType): string;
function TDistributionTypeFromStr(v: string): TDistributionType;
function ConvertToTSystemSize(pop: double): TSystemSize;

function Discount(const Yr : integer; const DiscRate, Value : double) : double;
function cDiscount(const Yr,DiscRateIdx : integer; const Value : double) : double;
procedure InitPreCalcDR(DiscRate: double);

procedure CSL(s: string);
function SetRandSeed(s : string) : integer;
function RemoveProxyLetter(s : string) : string;

implementation

procedure CSL(s: string);
begin
{$IFDEF DEBUG}
  CodeSite.Send(s);
{$ENDIF}
end;

function SetRandSeed(s : string) : integer;
var ss : string;
    i : integer;
begin
  ss:='';
  for i:=1 to length(s) do begin
    if s[i] in ['0'..'9'] then ss:=ss+s[i];
  end;
  while (ss[1] = '0') do Delete(ss,1,1);

  ss := stringreplace(ss, '0', '', [rfReplaceAll, rfIgnoreCase]);
  if length(ss) > 9 then ss := copy(ss, length(ss)-8, 9);

  result:=strtoint(ss);
end;

function RemoveProxyLetter(s : string) : string;
begin
  Result:=s;
  if Result[length(Result)] in ['a'..'z'] then Delete(Result,length(Result),1);
end;
```

```
function TSystemSizeToStr(size:TSystemSize): string;
begin
     if (size = ss0_100) then
          Result := 'ss0_100'
     else if (size = ss101_500) then
          Result := 'ss101_500'
     else if (size = ss501_1000) then
          Result := 'ss501_1000'
     else if (size = ss1001_3300) then
          Result := 'ss1001_3300'
     else if (size = ss3301_10k) then
          Result := 'ss3301_10k'
     else if (size = ss10k_50k) then
          Result := 'ss10k_50k'
     else if (size = ss50k_100k) then
          Result := 'ss50k_100k'
     else if (size = ss100k_1m) then
          Result := 'ss100k_1m'
     else if (size = ss1m) then
          Result := 'ss1m'
     else
          raise Exception.Create('Invalid TSystemSize of: ' + inttostr(ord(size)));
end;

function TSourceWaterToStr(source:TSourceWater): string;
begin
     if (source = swGroundWater) then
          Result := 'gw'
     else if (source = swSurfaceWater) then
          Result := 'sw'
     else
          raise Exception.Create('Invalid TSourceWater of: ' +
inttostr(ord(source)));
end;

function StrToTSystemSize(value:String): TSystemSize;
begin
     if (value = 'ss0_100') then
          Result := ss0_100
     else if (value = 'ss101_500') then
          Result := ss101_500
     else if (value = 'ss501_1000') then
          Result := ss501_1000
     else if (value = 'ss1001_3300') then
          Result := ss1001_3300
     else if (value = 'ss3301_10k') then
          Result := ss3301_10k
     else if (value = 'ss10k_50k') then
```

```
          Result := ss10k_50k
     else if (value = 'ss50k_100k') then
          Result := ss50k_100k
     else if (value = 'ss100k_1m') then
          Result := ss100k_1m
     else if (value = 'ss1m') then
          Result := ss1m
     else
          raise Exception.Create('Invalid System Size string: ' + value);
end;

function StrToTSourceWater(value:string): TSourceWater;
begin
     if (value = 'GW') then
          Result := swGroundWater
     else if (value = 'SW') then
          Result := swSurfaceWater
     else
          raise Exception.Create('Invalid Source Water string: ' + value);
end;

function StrToTOwnership(value:string): TOwnership;
begin
     if (value = 'Public') then
          Result := oPublic
     else if (value = 'Private') then
          Result := oPrivate
     else
          raise Exception.Create('Invalid Ownership string: ' + value);
end;

function StrToTSystemType(value:string): TSystemType;
begin
     if (value = 'CWS') then
          Result := sysCWS
     else if (value = 'NTNCWS') then
          Result := sysNTNC
     else if (value = 'TNCWS') then
          Result := sysTNC
     else
          raise Exception.Create('Invalid System Type string: ' + value);
end;

function TOwnershipToStr(owner:TOwnership): string;
begin
     if (owner = oPublic) then
          Result := 'Public'
     else if (owner = oPrivate) then
```

```
          Result := 'Private'
     else
          raise Exception.Create('Invalid TOwnership of: ' + inttostr(ord(owner)));
end;

function TSystemTypeToStr(systype:TSystemType) : string;
begin
     if (systype = sysCWS) then
          Result := 'sysCWS'
     else if (systype = sysNTNC) then
          Result := 'sysNTNCWS'
     else if (systype = sysTNC) then
          Result := 'sysTNC'
     else
          raise Exception.Create('Invalid TSystemType of: ' +
inttostr(ord(systype)));
end;

function IntToTAge(Age: Integer): TAge;
begin
     case Age of
          0..4:         Result := a0to4;
          5..9:         Result := a5to9;
          10..14:       Result := a10to14;
          15..19:       Result := a15to19;
          20..24:       Result := a20to24;
          25..29:       Result := a25to29;
          30..34:       Result := a30to34;
          35..39:       Result := a35to39;
          40..44:       Result := a40to44;
          45..49:       Result := a45to49;
          50..54:       Result := a50to54;
          55..59:       Result := a55to59;
          60..64:       Result := a60to64;
          65..69:       Result := a65to69;
          70..74:       Result := a70to74;
          75..79:       Result := a75to79;
          80..84:       Result := a80to84;
          85..MaxInt:   Result := a85U;
     else
         raise Exception.Create('FATAL ERROR - Age is out of range: ' +
IntToStr(Age));
     end;
end;

function TGenderToStr(Gender: TGender): string;
begin
     case Gender of
```

```
          gMale: result := 'Male';
          gFemale: result := 'Female';
     end;
end;

function TAgeToStr(Age: TAge): string;
begin
     case Age of
          a0to4: result := '0to4';
          a5to9: result := '5to9';
          a10to14: result := '10to14';
          a15to19: result := '15to19';
          a20to24: result := '20to24';
          a25to29: result := '25to29';
          a30to34: result := '30to34';
          a35to39: result := '35to39';
          a40to44: result := '40to44';
          a45to49: result := '45to49';
          a50to54: result := '50to54';
          a55to59: result := '55to59';
          a60to64: result := '60to64';
          a65to69: result := '65to69';
          a70to74: result := '70to74';
          a75to79: result := '75to79';
          a80to84: result := '80to84';
          a85U:    result := '85+';
     end;
end;

function StrSAToTState(StateString: string): TState;
begin
     if ( AnsiSameText(StateString,'Al') ) then
        Result := sAlabama
     else if ( AnsiSameText(StateString,'Ak') ) then
          Result := sAlaska
     else if ( AnsiSameText(StateString,'Az') ) then
          Result := sArizona
     else if ( AnsiSameText(StateString,'Ar') ) then
          Result := sArkansas
     else if ( AnsiSameText(StateString,'Ca') ) then
          Result := sCalifornia
     else if ( AnsiSameText(StateString,'Co') ) then
          Result := sColorado
     else if ( AnsiSameText(StateString,'Ct') ) then
          Result := sConnecticut
     else if ( AnsiSameText(StateString,'De') ) then
          Result := sDelaware
     else if ( AnsiSameText(StateString,'Dc') ) then
```

```
     Result := sDistrictOfColumbia
else if ( AnsiSameText(StateString,'Fl') ) then
     Result := sFlorida
else if ( AnsiSameText(StateString,'Ga') ) then
     Result := sGeorgia
else if ( AnsiSameText(StateString,'Hi') ) then
     Result := sHawaii
else if ( AnsiSameText(StateString,'Id') ) then
     Result := sIdaho
else if ( AnsiSameText(StateString,'Il') ) then
     Result := sIllinois
else if ( AnsiSameText(StateString,'In') ) then
     Result := sIndiana
else if ( AnsiSameText(StateString,'Ia') ) then
     Result := sIowa
else if ( AnsiSameText(StateString,'Ks') ) then
     Result := sKansas
else if ( AnsiSameText(StateString,'Ky') ) then
     Result := sKentucky
else if ( AnsiSameText(StateString,'La') ) then
     Result := sLouisiana
else if ( AnsiSameText(StateString,'Me') ) then
     Result := sMaine
else if ( AnsiSameText(StateString,'Md') ) then
     Result := sMaryland
else if ( AnsiSameText(StateString,'Ma') ) then
     Result := sMassachusetts
else if ( AnsiSameText(StateString,'Mi') ) then
     Result := sMichigan
else if ( AnsiSameText(StateString,'Mn') ) then
     Result := sMinnesota
else if ( AnsiSameText(StateString,'Ms') ) then
     Result := sMississippi
else if ( AnsiSameText(StateString,'Mo') ) then
     Result := sMissouri
else if ( AnsiSameText(StateString,'Mt') ) then
     Result := sMontana
else if ( AnsiSameText(StateString,'Ne') ) then
     Result := sNebraska
else if ( AnsiSameText(StateString,'Nv') ) then
     Result := sNevada
else if ( AnsiSameText(StateString,'Nh') ) then
     Result := sNewHampshire
else if ( AnsiSameText(StateString,'Nj') ) then
     Result := sNewJersey
else if ( AnsiSameText(StateString,'NM') ) then
     Result := sNewMexico
else if ( AnsiSameText(StateString,'NY') ) then
```

```
     Result := sNewYork
else if ( AnsiSameText(StateString,'Nc') ) then
     Result := sNorthCarolina
else if ( AnsiSameText(StateString,'Nd') ) then
     Result := sNorthDakota
else if ( AnsiSameText(StateString,'Oh') ) then
     Result := sOhio
else if ( AnsiSameText(StateString,'Ok') ) then
     Result := sOklahoma
else if ( AnsiSameText(StateString,'Or') ) then
     Result := sOregon
else if ( AnsiSameText(StateString,'Pa') ) then
     Result := sPennsylvania
else if ( AnsiSameText(StateString,'Ri') ) then
     Result := sRhodeIsland
else if ( AnsiSameText(StateString,'Sc') ) then
     Result := sSouthCarolina
else if ( AnsiSameText(StateString,'Sd') ) then
     Result := sSouthDakota
else if ( AnsiSameText(StateString,'Tn') ) then
     Result := sTennessee
else if ( AnsiSameText(StateString,'Tx') ) then
     Result := sTexas
else if ( AnsiSameText(StateString,'Ut') ) then
     Result := sUtah
else if ( AnsiSameText(StateString,'Vt') ) then
     Result := sVermont
else if ( AnsiSameText(StateString,'Va') ) then
     Result := sVirginia
else if ( AnsiSameText(StateString,'Wa') ) then
     Result := sWashington
else if ( AnsiSameText(StateString,'Wv') ) then
     Result := sWestVirginia
else if ( AnsiSameText(StateString,'Wi') ) then
     Result := sWisconsin
else if ( AnsiSameText(StateString,'Wy') ) then
     Result := sWyoming
else if ( AnsiSameText(StateString,'As') ) then
     Result := sAmericanSamoa
else if ( AnsiSameText(StateString,'Gu') ) then
     Result := sGuam
else if ( AnsiSameText(StateString,'MP') ) then
     Result := sNorthernMarianas
else if ( AnsiSameText(StateString,'PW') ) then
     Result := sPalau
else if ( AnsiSameText(StateString,'PR') ) then
     Result := sPuertoRico
else if ( AnsiSameText(StateString,'VI') ) then
```

```
          Result := sVirginIslands
     else if ( AnsiSameText(StateString,'01') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'02') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'03') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'04') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'05') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'06') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'07') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'08') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'09') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'10') ) then
          Result := sTribes
     else if ( AnsiSameText(StateString,'NN') ) then
          Result := sTribes
     else
          Result := sUnknown;
         //raise Exception.Create('Invalid State String: ' + StateString);
end;

function ConvertToTSystemSize(pop: double): TSystemSize;
begin
     if (pop <= 100) then
          Result := ss0_100
     else if (pop <= 500) then
          Result := ss101_500
     else if (pop <= 1000) then
          Result := ss501_1000
     else if (pop <= 3300) then
          Result := ss1001_3300
     else if (pop <= 10000) then
          Result := ss3301_10k
     else if (pop <= 50000) then
          Result := ss10k_50k
     else if (pop <= 100000) then
          Result := ss50k_100k
     else if (pop <= 1000000) then
          Result := ss100k_1m
     else
          Result := ss1m;
```

```
end;


procedure WriteStreamStr(Stream : TStream; Str : string);
var
  StrLen : integer;
begin
  StrLen := Length(Str) * SizeOf(Char);
  Stream.Write(StrLen, SizeOf(StrLen));
  if StrLen>0 then
    Stream.Write(Str[1], StrLen);
end;

function ReadStreamStr(Stream : TStream) : string;
var
  StrLen : integer;
begin
  Result := '';
  Stream.Read(StrLen,SizeOf(StrLen));
  if StrLen>0 then begin
    SetLength(Result, StrLen div SizeOf(Char));
    Stream.Read(Result[1], StrLen);
  end;
end;

{ TPicked }

procedure TPicked.SaveToStream(Strm : TStream);
begin
  WriteStreamStr(Strm,Name);
  Strm.Write(Size,sizeof(Size));
  Strm.Write(Owner,sizeof(Owner));
  Strm.Write(Region,sizeof(Region));
  Strm.Write(Source,sizeof(Source));
end;

procedure TPicked.LoadFromStream(Strm : TStream);
begin
  Name:=ReadStreamStr(Strm);
  Strm.Read(Size,sizeof(Size));
  Strm.Read(Owner,sizeof(Owner));
  Strm.Read(Region,sizeof(Region));
  Strm.Read(Source,sizeof(Source));
end;

function TPicked.GetFilter: string;
var F,s,r,o,sr,st : string;
    i,c : integer;
```

```
begin
  c:=0;
  s:='';
  for i:=1 to high(Size) do begin
    if Size[i] then begin
      if c>0 then
        S:=S+' or '+zSizeFilters[i]
      else
        S:=S+'('+zSizeFilters[i];
      inc(c);
    end;
  end;
  if c>0 then S:=S+')';

  c:=0;
  r:='';
  for i:=1 to high(Region) do begin
    if Region[i] then begin
      if c>0 then
        R:=R+' or '+zRegFilters[i]
      else
        R:=R+'('+zRegFilters[i];
      inc(c);
    end;
  end;
  if c>0 then R:=R+')';

  c:=0;
  o:='';
  for i:=1 to high(Owner) do begin
    if Owner[i] then begin
      if c>0 then
        o:=o+' or '+zOwnerFilters[i]
      else
        o:=o+'('+zOwnerFilters[i];
      inc(c);
    end;
  end;
  if c>0 then o:=o+')';

  c:=0;
  sr:='';
  for i:=1 to high(Source) do begin
    if Source[i] then begin
      if c>0 then
        sr:=sr+' or '+zSrcFilters[i]
      else
        sr:=sr+'('+zSrcFilters[i];
```

```
      inc(c);
    end;
  end;
  if c>0 then sr:=sr+')';

  c:=0;
  st:='';
  for i:=1 to high(SysType) do begin
    if SysType[i] then begin
      if c>0 then
        st:=st+' or '+zTypeFilters[i]
      else
        st:=st+'('+zTypeFilters[i];
      inc(c);
    end;
  end;
  if c>0 then st:=st+')';


  F:=S;
  if (F>'') and (R>'') then F:=F+' AND ' +r else if (R>'') then F:=R;
  if (F>'') and (O>'') then F:=F+' AND ' +o else if (o>'') then F:=o;
  if (F>'') and (SR>'') then F:=F+' AND ' +sr else if (sr>'') then F:=sr;
  if (F>'') and (ST>'') then F:=F+' AND ' +sT else if (st>'') then F:=st;
  Result:=F;
end;

constructor TPicked.create;
begin
  inherited;
  fillchar(size,sizeof(size),0);
  fillchar(region,sizeof(region),0);
  fillchar(owner,sizeof(owner),0);
  fillchar(source,sizeof(source),0);
  fillchar(systype,sizeof(systype),0);
end;

function TPicked.GetName: string;
var n : string;
    i : integer;
    procedure AddItN(aSt : string);
    begin
      if N>'' then N:=N+','+aST else N:=aST;
    end;
begin
  N:='';
  for i:=1 to high(Size) do
    if Size[i] then
```

```
      AddItN(zSizenames[i]);
  for i:=1 to high(Region) do
    if Region[i] then
      AddItN(zRegNames[i]);
  for i:=1 to high(Owner) do
    if Owner[i] then
      AddItN(zOwnernames[i]);
  for i:=1 to high(Source) do
    if Source[i] then
      AddItN(zSrcnames[i]);
  for i:=1 to high(SysType) do
    if SysType[i] then
      AddItN(zTypeNames[i]);

  Result:=N;
end;

// For reading Data Request database
function TDataNeedTypeToStr(v: TDataNeedType): string;
begin
     if (v = dnCost) then
          Result := 'Cost of material and/or operation'
     else if (v = dnCostFunction) then
          Result := 'Cost of material and/or operation Function'
     else if (v = dnCount) then
          Result := 'Count of Samples'
     else if (v = dnDemographic) then
          Result := 'Demographic'
     else if (v = dnDistribution) then
          Result := 'Distribution'
     else if (v = dnHours) then
          Result := 'Estimated Hours'
     else if (v = dnRate) then
          Result := 'Hourly Rate'
     else if (v = dnNumSamp) then
          Result := 'Number of Samples'
     else if (v = dnPercentage) then
          Result := 'Percentage'
     else if (v = dnProbability) then
          Result := 'Probability'
     else
          raise Exception.Create('Invalid TDataNeedType of: ' + inttostr(ord(v)));
end;

function TDataNeedTypeFromStr(v: string): TDataNeedType;
begin
     if (v = 'Cost of material and/or operation') then
          Result := dnCost
```

```
     else if (v = 'Cost of material and/or operation Function') then
          Result := dnCostFunction
     else if (v = 'Count of Samples') then
          Result := dnCount
     else if (v = 'Demographic') then
          Result := dnDemographic
     else if (v = 'Distribution') then
          Result := dnDistribution
     else if (v = 'Estimated Hours') then
          Result := dnHours
     else if (v = 'Hourly Rate') then
          Result := dnRate
     else if (v = 'Number of Samples') then
          Result := dnNumSamp
     else if (v = 'Percentage') then
          Result := dnPercentage
     else if (v = 'Probability') then
          Result := dnProbability
     else
          raise Exception.Create('Invalid DataNeedType of: ' + v);
end;

function TCostBaseToStr(v: TCostBase): string;
begin
     if (v = cbEP) then
          Result := 'EP'
     else if (v = cbState) then
          Result := 'State'
     else if (v = cbSystem) then
          Result := 'System'
     else
          raise Exception.Create('Invalid TCostBase of: ' + inttostr(ord(v)));
end;

function TCostBaseFromStr(v: string): TCostBase;
begin
     if (v = 'EP') then
          Result := cbEP
     else if (v = 'State') then
          Result := cbState
     else if (v = 'System') then
          Result := cbSystem
     else
          raise Exception.Create('Invalid CostBase of: ' + v);
end;

function TUnitBasisToStr(v: TUnitBasis): string;
begin
```

```
     if (v = ubOneTime) then
          Result := 'One-time'
     else if (v = ubPerYear) then
          Result := 'Per Year'
     else if (v = ubPerYearVariable) then
          Result := 'Per Year; Variable by Year'
     else if (v = ubSystem) then
          Result := 'System'
     else if (v = ubBlank) then
          Result := ''
     else
          raise Exception.Create('Invalid TUnitBasis of: ' + inttostr(ord(v)));
end;

function TUnitBasisFromStr(v: string): TUnitBasis;
begin
     if (v = 'One-time') then
          Result := ubOneTime
     else if (v = 'Per Year') then
          Result := ubPerYear
     else if (v = 'Per Year; Variable by Year') then
          Result := ubPerYearVariable
     else if (v = 'System') then
          Result := ubSystem
     else if (v = '') then
          Result := ubBlank
     else
          raise Exception.Create('Invalid UnitBasis of: ' + v);
end;

function TDistributionTypeToStr(v: TDistributionType): string;
begin
     if (v = dtPointEstimate) then
          Result := 'Point Estimate'
     else if (v = dtUniform) then
          Result := 'Uniform'
     else if (v = dtTriangular) then
          Result := 'Triangular'
     else if (v = dtNormal) then
          Result := 'Normal'
     else if (v = dtLogNormal) then
          Result := 'Log Normal'
     else if (v = dtFunction) then
          Result := 'Function'
     else if (v = dtCustom) then
          Result := 'Custom'
     else
          raise Exception.Create('Invalid TDistributionType of: ' +
```

```
inttostr(ord(v)));
end;

function TDistributionTypeFromStr(v: string): TDistributionType;
begin
     if (v = 'Point Estimate') then
          Result := dtPointEstimate
     else if (v = 'Uniform') then
          Result := dtUniform
     else if (v = 'Triangular') then
          Result := dtTriangular
     else if (v = 'Normal') then
          Result := dtNormal
     else if (v = 'Log Normal') then
          Result := dtLogNormal
     else if (v = 'Function') then
          Result := dtFunction
     else if (v = 'Custom') then
          Result := dtNormal
     else
          raise Exception.Create('Invalid DistributionType of: ' + v);
end;

{ TSafeWaterPercUncFile }

function FileLineCount(aFIle : string) : integer;
var T : TStringLIst;
begin
  T:=TStringLIst.Create;
  Result:=0;
  try
   T.LoadFromFile(aFile);
   Result:=T.Count;
  except
  end;
  T.Free;
end;

function ReadPerUncFile(aFIle : string; var Ps : array of TPercentiles) : boolean;
var T,TL : TStringLIst;
    IFile: textfile;
    S : string;
    i,j : integer;
begin
  assignfile(IFile,aFile);
  try
    reset(IFile);
  except
```

```
    Result:=False;
    exit;
  end;

  TL:=TStringLIst.Create;
  T:=TStringLIst.Create;
  try
    while not eof(IFile) do begin
      readln(IFile,S);
      T.Add(S);
    end;

    for i:=0 to T.Count-1 do begin
      TL.CommaText:=T[i];
      for j:=low(Ps[i]) to high (Ps[i]) do begin
        Ps[i,j]:=strtofloat(TL[j]);
      end;
    end;
    Result:=True;
  except
    Result:=False;
  end;

  closefile(IFile);
  TL.Free;
  T.Free;
end;

class function TSafeWaterPercUncFile.GetIX(aFile: string; aIX: integer;
  var P: TPercentiles) : double;
var Ps : array of TPercentiles;
    i,ix : integer;
    s : double;
begin
  fillchar(P,SizeOf(P),0);
  i:=FileLineCount(aFile);
  if i<=0 then exit;
  Setlength(Ps,i);
  if not ReadPerUncFile(aFile,Ps) then exit;
  if (aIX<0) then IX:=0 else
  if (aIX>High(PS)) then IX:=high(PS)
  else IX:=AIX;
  s:=0;
  for i:=low(P) to high(P) do begin
    P[i]:=Ps[Aix,i];
    s:=s+Ps[Aix,i];
  end;
  Result:=s/(high(P)+1);
```

```
end;

class function TSafeWaterPercUncFile.GetMean(aFile: string;
  var P: TPercentiles) : double;
var Ps : array of TPercentiles;
    i,j : integer;
begin
  Result:=TSafeWaterPercUncFile.GetIX(aFile,0,P);
  (*
  fillchar(P,SizeOf(P),0);
  i:=FileLineCount(aFile);
  if i<=0 then exit;
  Setlength(Ps,i);
  if not ReadPerUncFile(aFile,Ps) then exit;
  if not High(Ps)>=0 then exit;
  for i:=low(Ps) to High(Ps) do begin
    for j:=low(Ps[i]) to High(Ps[i]) do begin
      p[j]:=p[j]+Ps[i,j];
    end;
  end;
  for j:=low(P) to High(P) do begin
     p[j]:=p[j]/(High(Ps)+1);
  end;
  *)
end;

class function TSafeWaterPercUncFile.GetRandom(aFile: string;
  var P: TPercentiles) : double;
var Ps : array of TPercentiles;
    i : integer;
    aix : integer;
begin
  fillchar(P,SizeOf(P),0);
  i:=FileLineCount(aFile);
  if i<=0 then exit;
  Setlength(Ps,i);
  if not ReadPerUncFile(aFile,Ps) then exit;
  aix:=1+Random(High(Ps)-1);
  Result:=TSafeWaterPercUncFile.GetIX(aFile,aix,P);
end;

class function TSafeWaterPercUncFile.ValidFile(aFile: string): boolean;
var Ps : array of TPercentiles;
    i : integer;
begin
  i:=FileLineCount(aFile);
  Result:=False;
  if i>0 then begin
```

```
    Setlength(Ps,i);
    Result:=ReadPerUncFile(aFile,Ps);
    if Result then Result:=High(Ps)>=0;
  end;
end;

function Discount(const Yr : integer; const DiscRate, Value : double) : double;
begin
  Result:=value/intpower((1+DiscRate),Yr);
end;

function cDiscount(const Yr,DiscRateIdx : integer; const Value : double) : double;
begin
  //Result:=Value;
  Result:=value/PreCalcDiscRate[Yr];
end;

procedure InitPreCalcDR(DiscRate: double);
var i : integer;
begin
  for i:=low(PreCalcDiscRate) to high(PreCalcDiscRate) do
     PreCalcDiscRate[i]:=intpower((1+DiscRate),i);
end;

procedure SteupGMoveBinLC;
var f,tt : integer;
begin
  for f:=low(GMoveBinLC) to high(GMoveBinLC) do begin
    for tt:=low(GMoveBinLC[f]) to high(GMoveBinLC[f]) do begin
      GMoveBinLC[f,tt]:=0;
    end;
  end;
  //set up possible movements - must be a better way... 1 CCT, 2 - LSLR
  GMoveBinLC[1,13]:=1; GMoveBinLC[1,15]:=2; GMoveBinLC[1,16]:=2; GMoveBinLC[1,3]:=2;
  GMoveBinLC[2,14]:=1; GMoveBinLC[2,15]:=2; GMoveBinLC[2,16]:=2; GMoveBinLC[2,3]:=2;
  GMoveBinLC[3,15]:=1; GMoveBinLC[3,16]:=2;
  GMoveBinLC[4,13]:=1; GMoveBinLC[4,15]:=2; GMoveBinLC[4,16]:=2;
GMoveBinLC[4,10]:=2;
  GMoveBinLC[7,14]:=1; GMoveBinLC[7,15]:=2; GMoveBinLC[7,16]:=2;
GMoveBinLC[7,10]:=2;
  GMoveBinLC[10,15]:=1; GMoveBinLC[10,16]:=2;
  GMoveBinLC[13,15]:=2; GMoveBinLC[13,16]:=2;
  GMoveBinLC[14,15]:=2; GMoveBinLC[14,16]:=2;
  GMoveBinLC[15,16]:=2;
end;


{ TCostGenRec }
```

```
function TCostGenRec.DataString: string;
begin

result:=PWSid+','+SourceWater.toString+','+SystemSize.toString+','+Ownership.toStrin
g
    +','+SystemType.toString+','+
    EntryPoints.toString+','+AvgRevenue.toString+','+ PWSAnnualRevenue.toString+','
    +Population.toString+','+
    CostCapital.toString+','+SamplingWeight.toString+','+LSL.toString+','+
CCT.toString+','+Connections.toString+','+

NumProxies.toString+','+First_ale.toString+','+DFlowEP.toString+','+AFlowEP.toString
+','+NumberLSLs.toString+','+
    CCTPO4.toString+','+ CCTPH.toString+','+
CCTBoth.toString+','+BaselinePO4Dose.toString+','
    +Baselineph_wPh.toString+','+Baselineph_woPh.toString+','+
    Baselineph_woCCT.toString+','+ Baselineph_wPO4ph.toString+','+P90_base.toString;
end;

class function TCostGenRec.HeaderString: string;
begin
  result:='PWSid,SourceWater,SystemSize,Ownership,SystemType,'+
    'EntryPoints,AvgRevenue, PWSAnnualRevenue,Population,'+
    'CostCapital,SamplingWeight,LSL, CCT,Connections,'+
    'NumProxies,First_ale,DFlowEP,AFlowEP,NumberLSLs,'+
    'CCTPO4, CCTPH, CCTBoth,BaselinePO4Dose,Baselineph_wPh,Baselineph_woPh,'+
    'Baselineph_woCCT, Baselineph_wPO4ph,P90_base';
end;

initialization
  //WindowsVersion:=WinSystem;
  //ProcessorCount:=NumProcessors;
  //LogicalProcessors:=NumberLogicalProcessors;
  SteupGMoveBinLC;

  zSizeNames[1]:='Less than 100';
  zSizeFilters[1]:='(Population<=100)';
  zSizeNames[2]:='100 to 500';
  zSizeFilters[2]:='(Population>100) AND (Population<=500)';
  zSizeNames[3]:='500 to 1000';
  zSizeFilters[3]:='(Population>500) AND (Population<=1000)';
  zSizeNames[4]:='1000 to 3300';
  zSizeFilters[4]:='(Population>1000) AND (Population<=3300)';
  zSizeNames[5]:='3300 to 10000';
  zSizeFilters[5]:='(Population>3300) AND (Population<=10000)';
  zSizeNames[6]:='10000 to 50000';
  zSizeFilters[6]:='(Population>10000) AND (Population<=50000)';
```

```
  zSizeNames[7]:='50000 to 100000';
  zSizeFilters[7]:='(Population>50000) AND (Population<=100000)';
  zSizeNames[8]:='100000 to 1000000';
  zSizeFilters[8]:='(Population>100000) AND (Population<=1000000)';
  zSizeNames[9]:='Greater than 1000000';
  zSizeFilters[9]:='Population>1000000';


  zOwnerNames[1]:='Private';
  zOwnerFilters[1]:='(Ownership='+AnsiQuotedStr('Private',#39)+')';
  zOwnerNames[2]:='Public';
  zOwnerFilters[2]:='(Ownership='+AnsiQuotedStr('Public',#39)+')';

  zSrcNames[1]:='Ground';
  zsrcFilters[1]:='(SourceWater='+AnsiQuotedStr('GW',#39)+')';
  zSrcNames[2]:='Surface';
  zsrcFilters[2]:='(SourceWater='+AnsiQuotedStr('SW',#39)+')';

  zTypeNames[1]:='CWS';
  zTypeFilters[1]:='(SystemType='+AnsiQuotedStr('CWS',#39)+')';
  zTypeNames[2]:='NTNCWS';
  zTypeFilters[2]:='(SystemType='+AnsiQuotedStr('NTNCWS',#39)+')';

{
  Region labels changed 9/22/11
  zRegNames[1]:='1 - New England';
  zRegFilters[1]:='(EPARegion=1)';
  zRegNames[2]:=' 2 -  NJ, NY, PR, VI, & 7 Tribes';
  zRegFilters[2]:='(EPARegion=2)';
  zRegNames[3]:=' 3 - Mid-Atlantic';
  zRegFilters[3]:='(EPARegion=3)';
  zRegNames[4]:=' 4 - Southeast';
  zRegFilters[4]:='(EPARegion=4)';
  zRegNames[5]:=' 5 - IL, IN, MI, MN, OH, WI, & 35 Tribes';
  zRegFilters[5]:='(EPARegion=5)';
  zRegNames[6]:=' 6 - South Central';
  zRegFilters[6]:='(EPARegion=6)';
  zRegNames[7]:=' 7 - IA, KS, MO, NE, & 9 Tribes';
  zRegFilters[7]:='(EPARegion=7)';
  zRegNames[8]:=' 8 - Mountains & Plains';
  zRegFilters[8]:='(EPARegion=8)';
  zRegNames[9]:=' 9 - Pacific Southwest';
  zRegFilters[9]:='(EPARegion=9)';
  zRegNames[10]:='10 - Pacific Northwest';
  zRegFilters[10]:='(EPARegion=10)';
  zRegNames[11]:='99 - Other/Unknown';
  zRegFilters[11]:='(EPARegion=0)';
}
```

```
  zRegNames[1]:=' 1 - Region 1';
  zRegFilters[1]:='(EPARegion=1)';
  zRegNames[2]:=' 2 - Region 2';
  zRegFilters[2]:='(EPARegion=2)';
  zRegNames[3]:=' 3 - Region 3';
  zRegFilters[3]:='(EPARegion=3)';
  zRegNames[4]:=' 4 - Region 4';
  zRegFilters[4]:='(EPARegion=4)';
  zRegNames[5]:=' 5 - Region 5';
  zRegFilters[5]:='(EPARegion=5)';
  zRegNames[6]:=' 6 - Region 6';
  zRegFilters[6]:='(EPARegion=6)';
  zRegNames[7]:=' 7 - Region 7';
  zRegFilters[7]:='(EPARegion=7)';
  zRegNames[8]:=' 8 - Region 8';
  zRegFilters[8]:='(EPARegion=8)';
  zRegNames[9]:=' 9 - Region 9';
  zRegFilters[9]:='(EPARegion=9)';
  zRegNames[10]:='10 - Region 10';
  zRegFilters[10]:='(EPARegion=10)';

  //NewID:=0;
  StateNames:=TStringList.create;
  StateNames.Sorted:=True;
  StateNames.CommaText:=StateCommaList;


  {$ifdef DEBUGSAS}
    _BenIX:=_DH.AddCategory('Benefits');
    _CostIX:=_DH.AddCategory('Costs');
  {$endif}


finalization
  StateNames.Free;
end.
```