Programmer's Manual for GEMEDIT: An ActiveX Server for Integrating Finite Difference Modeling and Geographic Information Systems using INI-Style Files

Greg Pouch
Kansas Geological Survey

Abstract

Finite difference models are often the method of choice to model fluid flow. Such models require two-dimensional arrays of input parameters for each model cell as well as input data and options that are specific to a particular program. The arrays of input parameters, such as permeability and withdrawal through wells, have geographic meaning and it is desirable to be able to use GISs to manage the information and view the results.

Traditionally, the input arrays were obtained from hand-edited text files, by using a dedicated pre-/post-processor, or by using a set of transfer programs specific to the combination of GIS and model. The Grid Exchange Method (GEM) improves on this situation by providing a common format that can be used by any GIS and by any finite-difference model. This divides the problem into two much more tractable parts: for the GIS or GIS helper-program, reading and writing the arrays from and to the GEM file; for the model or model helper program, reading and writing the arrays from and to the GEM file.

This paper describes a server program, GEMEDIT, which can be called by other programs through an OLE/ActiveX interface. GEMEDIT provides basic utility functions to read and write various parts of a GEM file under Microsoft Windows. This paper is intended for use by programmer's developing Model-helper program or GIS-helper programs that use the GEM specification for exchanging finite difference model arrays. GEMEDIT can be used to edit GEM files, but is designed primarily as a component for use by other programs.

The specification of the Grid Exchange Method and includes information on how a GEM file is organized and what information it contains and how are described in a separate paper. A translation program that lets GEM files be used with the Windows-based Hydrogeologic Exploration and Appraisal Toolkit (WHEAT) Electronic Mapping programs is available and is described in a separate paper, although example source code from GEM_WHT is included here.

Table of Contents

Overview of the Grid Exchange Method

Each finite difference model requires input in its own particular format. Preparing the input datasets and reading the output results can require as much effort as the actual modeling of the system. Such a situation significantly impedes productivity and hinders modeling. The GEM file standard used herein solves this problem by breaking one large problem into a small number of tractable pieces that can be easily modified to take advantage of new models, new model features, and new GISs. It does this by storing model information in an easily-read Windows INI-style file.

GEM uses Windows INI-style files for two reasons: they are easy for humans to read and edit and they are easy for computer programs to read and edit. Although somewhat slower than a traditional dedicated file format or a database, this compromise file type allows more flexibility, extendibility, and portability. The analyst can hand-edit the file and not have to keep track of line and column number—a major task in a file containing only numbers for property values and program options. Computer programs can rapidly extract only the portions of interest to them and not be confused by comment-lines, line labels, and extra information used by the analyst or other programs.

The GEM file is a derivative of the standard MSWindows initialization (INI) file, a labeled, ascii file. An INI file is organized into sections and entries within sections. Sections are enclosed in brackets. Each entry follows the format EntryName=EntryValue. Semi-colons should be interpreted as comment characters.

The following is an example from my Windows.INI file

```
[Desktop]
IconTitleFaceName=MS Sans Serif
IconSpacing=75
Pattern=(None)
TileWallPaper=0
GridGranularity=0
WallPaper=(None)
IconTitleStyle=0
IconTitleSize=8

[Extensions]
crd=cardfile.exe ^.crd
trm=terminal.exe ^.trm
txt=notepad.exe ^.txt
ini=notepad.exe ^.ini
pcx=C:\MSOFFICE\CORELDRW\PHOTOPNT\corelpnt.exe ^.pcx
bmp=pbrush.exe ^.bmp
```

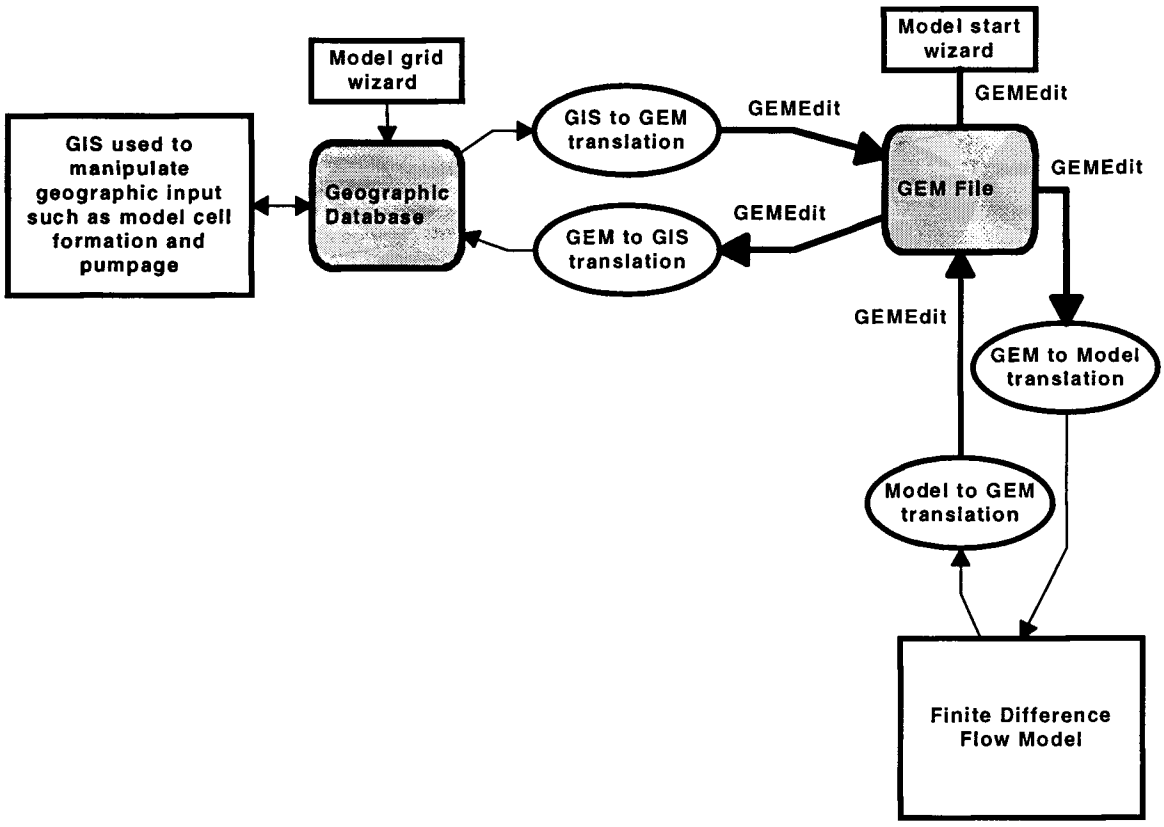
This shows two sections [Desktop] and [Extensions]. The first entry in [Desktop]

```
IconTitleFaceName=MS Sans Serif
```

indicates that icons use the MS Sans Serif typeface. The entry name is IconTitleFaceName, and the entry value is MS Sans Serif. Reading an INI file requires knowing the meaning of the sections and the entries. Guidelines for using an INI file to store the data needed for finite difference modeling are contained in the GEM Specification. An example GEM file is included at the end of this document.

This paper describes GEMEDIT, an ActiveX/OLE server component that can be used to read and write GEM files that can be used as a component by GIS-helper programs or by model-helper programs. In order to use GEMEDIT as a component, the programming language must be able to use ActiveX/OLE server components. Examples include Microsoft Visual Basic, Excel, Access, and Visual C++, and Borland C++ and

Delphi. If you can control Excel using the OLE interface, you can use GEMEDIT to manipulate GEM files. Alternatively, you can use the GEM specification and directly manipulate the GEM file, since it is a simple ascii text format.



GEMEDIT

Although the GEM file specification itself provides a basis for designing programs to use a GIS as a pre-processor/post-processor for a wide variety of finite difference models, software that actually implements this idea is even better. GEMEDIT is a MSWindows program written in Visual Basic 4.0 that 1) allows the user to manipulate arrays in a GEM file interactively with spreadsheet-style editor and 2) can be used as a component in a larger program that deals with GEM files, such as GEM_WHT, that handles GEM and WHEAT conversions.

GEMEDIT is an OLE server that exposes methods for creating a new GEM file, reading and writing arrays, reading and writing property mappings, reading and writing individual entries in an INI file, obtaining a list of arrays or sections in a GEM file. GEMEDIT can be used from other OLE-using programs in the same way as Excel, by declaring or creating an object of type GEMFile and using its properties and methods to perform tasks

In the following example, assume that the array Perm( ) contains permeabilities to be written to section [Permeability for Layer 1] and that much suggested data will not be written for clarity reasons.

```
Dim myGEMFile as new GEMFile
Dim aVar as variant
aVar=Perm( ) 'Copy the array in the variant
jErrCode = myMITIFile.FileOpen(strFileName:=strINIFile, nRows:=nRows,
nColumns:=nCols, nLayers:=nLayers)
myMITIFile.WriteArray Section:="Permeability for Layer 1",
VariantToHoldArray:=aVar, DDataType:="REAL"
```

Methods and Functions Available in GEMEdit

GEMEDIT can be used for two purposes: the user can directly edit a GEM file using it and it can be used as an OLE server providing access to GEM files to other programs, such as Model-GEM and GIS-GEM link programs.

This section explains the methods and functions available in GEMEdit as an OLE server. Data types are those used by Visual Basic 4.0. Most functions return an error code. For functions that return an error code, 0 indicates success. For functions that return a count, positive values indicate success and negative values are error codes.

Functions are listed in decreasing order of likelihood of use.

FILENEW

Used to create a new GEM file. All parameters are input. Fails if the file already exists, so this needs to be checked by the calling program. Returns an error code.

```
Function FileNew(FileName As String, nColumns As Long, nRows As Long, nLayers As
Long) As Long
```

FILEOPEN

Used to open a pre-existing GEM file. FileName is input, others are output. Fails if the file does not already exist, so this needs to be checked by the calling program. Returns an error code.

```
Function FileOpen(FileName As String, nColumns As Long, nRows As Long, nLayers As
Long) As Long
```

ARRAYNAMES

Used to list array sections. If Index is specified, it returns the name of that array. If not specified, it returns a variant containing an array of strings.

```
Function ArrayNames(Optional Index As Variant) As Variant
```

READARRAY

Reads an array from the file. Section is the name of the desired array. Output is a variant that will hold an array of the appropriate type and a text string describing the datatype. Specifying optional variables will be filled with the appropriate data on output, such as a VariableName. Return value is an error code.

```
Function ReadArray(Section As String, VariantToHoldArray As Variant, DataType As
String, Optional Units, Optional CommentsForUser, Optional VariableName, Optional
MissingValuesShownAs, Optional RevisionNumber, Optional Accuracy, Optional
LastModfiedTime, Optional LastModfiedAuthor) As Long
```

READARRAYPROPERTIES

Reads everything in an array section that isn't part of the array. Used for providing comments, variable name, units information and such to other programs. The return value is the number of properties returned if positive and an error code if negative. PropertyNames and PropertyValues hold the actual answers on output

```
Function ReadArrayProperties(Section As String, PropertyNames() As String,
PropertyValues() As Variant) As Long
```

WRITEARRAY

Used to write an array section. The array itself is stored in a variant. All parameters are output. Return value is an error code.

```
Function WriteArray(Section As String, VariantToHoldArray As Variant, DataType As
String, Optional Units, Optional CommentsForUser, Optional VariableName,
Optional MissingValuesShownAs, Optional RevisionNumber, Optional Accuracy,
Optional LastModfiedTime, Optional LastModfiedAuthor) As Long
```

APPENDARRAYTOFILE

Writes the data in the specified section to the file in the format specified. The format should be a standard Visual Basic format string (same as an Excel Text format string or an Access format string). The file must be closed for this to work. This might be called several times in succession. Remember to erase or re-name an old input file before using this to construct a new input file. Return value is an error code.

```
Public Function AppendArrayToFile(Section As String, FileName As String,
OutputFormat As String) As Long
```

READENTRY

Used to read a single entry in the file. Can be replaced by the corresponding function in MegaINI. Do not use the Windows API functions if the file is larger than 64KB.

```
Function ReadEntry(Section As String, Entry As String, DefaultValue As String) As
String
```

WRITEENTRY

Used to write a single entry in the file. Can be replaced by the corresponding function in MegaINI. Do not use the Windows API functions if the file is larger than 64KB.

```
Sub WriteEntry(Section As String, Entry As String, YourMessageHere As String)
```

PROPERTYMAPPINGNAMES

Used to list PropertyMappings. Same usage as ArrayNames

```
Function PropertyMappingNames(Optional Index As Variant) As Variant
```

READPROPERTYMAPPING

Used to read a property mapping. The return value is number of types if positive, an error code if negative. VariantToHoldNamesArray is an array that lists the types, such as unit names, and VariantToHoldValuesArray is a variant holding an array that lists the values, such as permeability.

```
Function ReadPropertyMapping(Section As String, VariantToHoldNamesArray As
Variant, VariantToHoldValuesArray As Variant, DataType As String, DefaultValue As
Variant, Units As Variant, VariableName As Variant, Optional CommentsForUser,
Optional RevisionNumber, Optional LastModfiedTime, Optional LastModfiedAuthor) As
Long
```

WRITEPROPERTYMAPPING

Used to write a property mapping. Usage is similar to ReadPropertyMapping

```
Function WritePropertyMapping(Section As String, VariantToHoldNamesArray As
Variant, VariantToHoldValuesArray As Variant, DataType As String, DefaultValue As
Variant, Units As Variant, VariableName As Variant, Optional CommentsForUser,
Optional RevisionNumber, Optional LastModfiedTime, Optional LastModfiedAuthor) As
Long
```

SECTIONNAMES

Used to list all Sections in the file, including Arrays and PropertyMappings Same usage as ArrayNames.

```
Public Function SectionNames(Optional Index As Variant) As Variant
```

REFRESH

Forces a read of the file and optionally re-builds the lists of array sections and PropertyMapping sections. Only useful if other programs might be manipulating the file

```
Sub Refresh(Optional RefreshTheFile)
```

Alternatively, you might want to use z__RefreshListOfArraysNoMatterWhat

Z__REFRESHLISTOFARRAYSNOMATTERWHAT

As the name suggests, blindly updates the lists of array, section, and property mappings.

```
z__RefreshListOfArraysNoMatterWhat
```

Z_UPDATELISTSOFARRAYSSECTIONSANDPROPS

Same as above, but does not update the time stamp.

```
z_UpdateListsOfArraysSectionsAndProps
```

Sample GEM File

```
[Introduction]
NumberOfColumns=10
NumberOfRows=3
CreatedAt=7/12/96 12:44:43 PM
NumberOfLayers=2

[Permeability Layer 1]
StorageStyle=RowYYYYYY
Row000001=1.1 0 0 4.1 5.1 0 0 8.1 0 0
Row000002=0 0 0 4.2 0 0 7.2 0 0 0
Row000003=1.3 0 0 0 0 6.3 0 0 0 10.3
DDataType=REAL
RevisionNumber=7
LastModfiedTime=7/24/96 4:03:48 PM
ThisIsAnArraySection=TRUE

[Permeability Layer 2]
ThisIsAnArraySection=TRUE
StorageStyle=RowYYYYYY
Row000001=1.1 1.2 0 0 0.009 0 0 0 0 0
Row000002=1.2 1.2E-07 0 4.2 0 0 0 0 9.2 0
Row000003=1.3 0.012 0 0 0 0 0 0 0 10.3
DDataType=REAL
RevisionNumber=4
LastModfiedTime=7/24/96 4:03:55 PM

[ModelCellSerialNumbers]
ThisIsAnArraySection=TRUE
StorageStyle=RowYYYYYY
Units=Dimensionless|1|
Row000001=1 2 3 4 5 6 7 8 9 10
Row000002=11 12 13 14 15 16 17 18 19 20
Row000003=21 22 23 24 25 26 27 28 39 40
DDataType=INTEGER
CommentForUser=Do not modify these values. They are needed by WHEAT.
RevisionNumber=4
LastModfiedTime=7/24/96 4:03:59 PM

[Lithology Layer 1]
ThisIsAnArraySection=TRUE
StorageStyle=RowYYYYYY
Units=Dimensionless|1|
Row000001=Kgh Kgh Kgh To To To Kgh Kgh Kgh To
Row000002=Kgh Kgh Kgh To To Kgh Kgh Kgh Kgh Kgh
Row000003=Kgh Kgh Kgh Kgh To To Kgh Kgh Kgh Kgh
DDataType=TEXT
CommentForUser=Formation codes in layer 1, based on cell center.
RevisionNumber=1
LastModfiedTime=7/24/96 4:08:35 PM

[LithologyToPermeabilityMapping]
ThisIsAPropertyMapping=true
DDataType=REAL
Units=ft/day|3.527777777778e-6|m^+1 s^-1
Members=To Kgh Kgr Kd Kch Kk
To=21
Kgh=0.01
Kgr=0.0001
Kd=5
Kch=2.5
Kk=0.001
DefaultValue=-1e20
VariableName=Hydraulic Conductivity

[List Of Arrays]
Array1=Permeability Layer 1
Array2=Permeability Layer 2
Array3=ModelCellSerialNumbers
Array4=Lithology Layer 1
```

Sample Code from GEM_WHT

COPYFROMGEMTOWHEAT

```
Public Function CopyFromGEMToWHEAT(strGEMFileName As String, _
                                   strSections() As String, strTableName As
String, _
                                   strDBName As String, strModelName As String)
As Long

'Given an input GEM file, this will copy all the arrays [sections]  listed
'  into fields in the output table strTableName in database strDBName

Dim mRows As Long, mColumns As Long, mLayers As Long, _
    nR As Long, nC As Long, nL As Long, n As Long
Dim strTemp As String, strSerNum As String
Dim TempVar As Variant
Dim db As Database, rcs As Recordset
Dim TempGEMFile As New GEMFile

On Error Resume Next
TempGEMFile.FileOpen filename:=strGEMFileName, nColumns:=mColumns, nRows:=mRows,
nLayers:=mLayers
'Get the serial name of this modelgrid

If strModelName = "" Then
   strModelName = TempGEMFile.ReadEntry(Section:="GEM_Wheat", Entry:="ModelGrid",
DefaultValue:="")
   If strModelName = "" Then Exit Function 'raise an error
End If
strSerNum = "ModelCellSerialNumber_" & strModelName

If strDBName = "" Then
   strDBName = TempGEMFile.ReadEntry(Section:="GEM_Wheat", Entry:="Database",
DefaultValue:="")
     If strDBName = "" Then Exit Function  'raise an error
End If

Set db = OpenDatabase(strDBName, False, False)
If Err <> 0 Then
    'raise an error
    MsgBox "Error #" & Err & vbCrLf & Error & vbCrLf & "Trying to open the
database file"
    Exit Function
End If

'Get the serial numbers from the model grid into an array
Set rcs = db.OpenRecordset("Select * FROM [" & strModelName & "_ModelGrid] ORDER
BY  Column, Row ;")
rcs.MoveFirst: rcs.MoveLast: rcs.MoveFirst
If rcs.RecordCount <> mRows * mColumns Then
    'raise an error
    Exit Function
End If

Dim fldSerNum As Field
Set fldSerNum = rcs(strSerNum)
Dim lSerNum() As Long
ReDim lSerNum(1 To mRows, 1 To mColumns)
For nC = 1 To mColumns
    For nR = 1 To mRows
        lSerNum(nR, nC) = fldSerNum ' rcs("ModelCellSerialNumber_" & strModelName)
        rcs.MoveNext
    Next nR
Next nC
rcs.Close: Set rcs = Nothing


'Next, copy the data into arrays hidden in variants
Dim avColumns() As Variant 'Each element of avColumns corresponds to a
[Section]/Column
Dim vArray As Variant, sDataType As String
```

```
ReDim avColumns(LBound(strSections) To UBound(strSections))

For n = LBound(strSections) To UBound(strSections)
    TempGEMFile.ReadArray Section:=strSections(n), VariantToHoldArray:=vArray,
DataType:=sDataType
    avColumns(n) = Array(strSections(n), vArray, sDataType)
Next n


'Now, construct a table to hold this data.
Dim tdf As TableDef, fld As Field
Dim sName As String, iType As Long

Set tdf = db.CreateTableDef(strTableName)
   'Cell serial number field
    Set fld = tdf.CreateField("ModelCellSerialNumber_" & strModelName, dbLong):
tdf.Fields.Append fld: Set fld = Nothing

    For n = LBound(strSections) To UBound(strSections)
        sName = Trim$(avColumns(n)(1))
        If sName = strSerNum Then GoTo LBL_SkipField
        If sName = "" Then GoTo LBL_SkipField
        Select Case avColumns(n)(3)
            Case "INTEGER", "LONG"
                Set fld = tdf.CreateField(sName, dbLong): tdf.Fields.Append fld:
Set fld = Nothing
            Case "REAL", "FLOAT", "SINGLE", "DOUBLE": iType = dbSingle
                Set fld = tdf.CreateField(sName, dbSingle): tdf.Fields.Append fld:
Set fld = Nothing
            Case Else
                Set fld = tdf.CreateField(sName, dbText, 255): tdf.Fields.Append
fld: Set fld = Nothing
        End Select
        TempVar = avColumns(n)(2)
        avColumns(n) = TempVar
        TempVar = Null
        'avColumns(n) = avColumns(n)(2)
LBL_SkipField:
    Next n
db.TableDefs.Append tdf
Set tdf = Nothing: db.TableDefs.Refresh

Set rcs = db.OpenRecordset(strTableName)
'Now, write the data out.
For nC = 1 To mColumns
    For nR = 1 To mRows
       DoEvents
       rcs.AddNew
        rcs("ModelCellSerialNumber_" & strModelName) = lSerNum(nR, nC)
        For n = LBound(strSections) To UBound(strSections)
            rcs(strSections(n)) = avColumns(n)(nR, nC)
        Next n
       rcs.Update
       DoEvents
    Next nR
Next nC
rcs.Close: Set rcs = Nothing

'At this point, we should add all the extra information as properties of the
field.
Dim nProps As Long, vPropNames As Variant, vPropValues As Variant, nP As Long

Set tdf = db.TableDefs(strTableName)
'Set TempGEMFile = Nothing
'TempGEMFile.FileOpen filename:=strGEMFileName, nColumns:=mColumns, nRows:=mRows,
nLayers:=mLayers

Dim sPropNames() As String, sPropValues() As String
ReDim sPropNames(0 To 0), sPropValues(0 To 0)

 'For some reason, adding properties this way causes ACCESS to crash when you
 '  try opening the table in design view.  Big crash.  Very big crash.
 ' Anyway, it does seem to really be there, it's just that it completely crashes
Access.
 'This causes a crash in GEMEdit when it does the GetEntries_All bit.
```

```
 'How to debug this???  I would need to have both GEMEdit and MegaINI running in
design mode, and I can only provide one instance of MegaINI in design mode.....
            'dim vNames as variant, vValues as variant
   For n = LBound(strSections) To UBound(strSections)
       strTemp = strSections(n)
       nProps = TempGEMFile.ReadArrayProperties(strTemp, sPropNames(),
sPropValues())
'       nProps = TempGEMFile.ReadArrayPropertiesVariant(Section:=strSections(n),
VariantToHoldPropertyNames:=vPropNames, VariantToHoldPropertyValues:=vPropValues)
       If nProps > 0 Then
         Set fld = tdf.Fields(strSections(n))
         vPropNames = sPropNames()
         vPropValues = sPropValues()
         Call SetFieldProperties(fld, vPropNames, vPropValues)
         Set fld = Nothing
       End If
   Next n

'Add a primary key on the cell serial number
    Dim idx As Index
    Set idx = tdf.CreateIndex("PrimaryKey")
     Set fld = idx.CreateField("ModelCellSerialNumber_" & strModelName)
     idx.Fields.Append fld: Set fld = Nothing
     idx.Primary = True
     tdf.Indexes.Append idx
     Set idx = Nothing
Set tdf = Nothing
Call AddToModelPropertiesList(strDBName:=strDBName, strTable:=strTableName,
strModelGrid:=strModelName, strVersionName:=strGEMFileName & Format$(Now,
"_YYMMDD_hhmm"), strUserComments:="Generated from" & strGEMFileName)
Set TempGEMFile = Nothing
db.Close: Set db = Nothing
End Function
```

COPYFROMWHEATTOGEM

```
Public Function CopyFromWheatToGEM(strDBName As String, strTable As String, _
                                   strModelGrid As String, _
                                   FieldsOut() As String, strGEMFileOut As
String) As Long

'This function copies the fields list in strFieldsOut() in the table strTableNaem
'  corresponding to model strModelGrid in database strDBName into the GEM File
specified as strGEMFileOut
'If the file strGEMFileOut does not exist, it will be created.  If it does exist,
it will be modified.
Dim myINISection As New INISection
Dim mRows As Long, mColumns As Long, mLayers As Long, lColumns As Long, lRows As
Long, lLayers As Long, _
    nR As Long, nC As Long, nL As Long, n As Long
Dim strTemp As String, strSQL As String, strSerNum As String
Dim db As Database, rcs As Recordset, qdf As QueryDef
Dim TempGEMFile As New GEMFile

On Error Resume Next

If strDBName = "" Or strTable = "" Or strModelGrid = "" Or strGEMFileOut = "" Then
     'raise an error
    Exit Function
End If
Set db = OpenDatabase(strDBName, False, False)
If Err <> 0 Then
    'raise an error
    Exit Function
End If

'Copy non-blank strings over to a an output list.
   Dim strFieldsOut() As String
   Dim nFields As Long
   nFields = UBound(FieldsOut) - LBound(FieldsOut) + 1
```

```
   ReDim strFieldsOut(1 To nFields)
   nL = nFields
   nFields = 0
   For n = 1 To nL
      If Trim$(FieldsOut(n)) <> "" Then
         nFields = nFields + 1
         strFieldsOut(nFields) = Trim$(FieldsOut(n))
      End If
   Next n
   If nFields = 0 Then Exit Function
   ReDim Preserve strFieldsOut(1 To nFields)



GoSub SUB_GetDimensions 'Get information on the model grid
GoSub SUB_CheckDimensions 'See if the GEM file already has model data in it.  If
it does and the model dimensions don't match this file, raise and error and exit
GoSub SUB_CheckGEM_WHT

'Start the query that gets cell data.  This must be ordered by column and row
strSerNum = "[ModelCellSerialNumber_" & strModelGrid & "]"
'SELECT DISTINCTROW Grid.Column, Grid.Row, Properties.*
'FROM LowerRattlesnake_ModelGrid AS Grid INNER JOIN
LowerRattlesnake_HighK_960730_1750 AS Properties ON
Grid.ModelCellSerialNumber_LowerRattlesnake =
Properties.ModelCellSerialNumber_LowerRattlesnake
'ORDER BY Grid.Column, Grid.Row;
strSQL = "SELECT DISTINCTROW Grid.Column, Grid.Row, Properties.* " & _
         " FROM [" & strModelGrid & "_ModelGrid] AS Grid INNER JOIN [" & strTable
& "] AS Properties " & _
         " ON Grid." & strSerNum & " = Properties." & strSerNum & " ORDER BY
Grid.Column, Grid.Row; "
Set rcs = db.OpenRecordset(strSQL)
rcs.MoveFirst: rcs.MoveLast: rcs.MoveFirst
If rcs.RecordCount <> mRows * mColumns Then
  'raise an error
  Exit Function
End If
'At this point, we have a valid GEM file with a GEM_Wheat section

TempGEMFile.FileOpen filename:=strGEMFileOut, nColumns:=lColumns, nRows:=lRows,
nLayers:=lLayers

Dim nFieldsOut As Long
nFieldsOut = UBound(strFieldsOut) - LBound(strFieldsOut) + 1
Dim vPropArrays() As Variant, vOnePropArray() As Variant, sDataType() As String
ReDim vPropArrays(LBound(strFieldsOut) To UBound(strFieldsOut)),
sDataType(LBound(strFieldsOut) To UBound(strFieldsOut))

    ReDim vOnePropArray(1 To mRows, 1 To mColumns)
For n = LBound(strFieldsOut) To UBound(strFieldsOut)
'    Debug.Print n, strFieldsOut(n), Now
'    Debug.Print n, rcs(strFieldsOut(n)).Name, rcs(strFieldsOut(n)).Type
    vPropArrays(n) = vOnePropArray()  'fill the array vPropArrays with copies of a
variant array with the right dimension
    Select Case rcs(strFieldsOut(n)).Type 'Set the data types for later use
        Case dbByte, dbInteger, dbLong
            sDataType(n) = "INTEGER"
        Case dbSingle, dbDouble
            sDataType(n) = "REAL"
        Case Else
            sDataType(n) = "TEXT"
    End Select
Next n


'Read the cell values in
For nC = 1 To mColumns
    For nR = 1 To mRows
        For n = LBound(strFieldsOut) To UBound(strFieldsOut)
            vPropArrays(n)(nR, nC) = rcs(strFieldsOut(n))
        Next n
        rcs.MoveNext
    Next nR
```

```
Next nC
Dim sPropNames() As String, vPropValues() As Variant, fld As Field, nP As Long
'Write the values out as sections
For n = LBound(strFieldsOut) To UBound(strFieldsOut)
    'Copy all the field properties to the section as entries
    Set fld = rcs(strFieldsOut(n))
    For nP = 1 To GetFieldProperties(fld, sPropNames(), vPropValues())
        'Do this first so that automatically-updated entries are not over-written
        TempGEMFile.WriteEntry Section:=strFieldsOut(n), Entry:=sPropNames(nP),
YourMessageHere:=CStr(vPropValues(nP))
    Next nP
    TempGEMFile.WriteArray Section:=strFieldsOut(n),
VariantToHoldArray:=vPropArrays(n), DataType:=sDataType(n)
Next n
TempGEMFile.z__RefreshListOfArraysNoMatterWhat
Set TempGEMFile = Nothing

Exit Function


SUB_CheckDimensions:
'Check model dimensions, and potentially leave the function if they don't match
  myINISection.filename = strGEMFileOut
  myINISection.SectionName = "Introduction"

   'Call PrivIniRegister(sSectionName:="Introduction",
sIniFileName:=strGEMFileOut)
    lColumns = myINISection.GetEntry(EntryName:="NumberOfColumns",
DefaultValue:="0")
    If lColumns = 0 Then
        myINISection.SetEntry EntryName:="NumberOfColumns",
EntryValue:=CStr(mColumns)
       Else
        If lColumns <> mColumns Then
            'Raise and error and leave
            Exit Function
        End If
    End If

    lRows = myINISection.GetEntry(EntryName:="NumberOfRows", DefaultValue:="0")
    If lRows = 0 Then
        myINISection.SetEntry EntryName:="NumberOfRows", EntryValue:=CStr(mRows)
       Else
        If lRows <> mRows Then
            'Raise and error and leave
            Exit Function
        End If
    End If

    lLayers = myINISection.GetEntry(EntryName:="NumberOfLayers",
DefaultValue:="0")
    If lLayers = 0 Then
        myINISection.SetEntry EntryName:="NumberOfLayers",
EntryValue:=CStr(mLayers)
       Else
        If lLayers <> mLayers Then
            'Raise and error and leave
            Exit Function
        End If
    End If
Return


SUB_GetDimensions:
    Set qdf = db.OpenQueryDef("zWHEAT_Models_GridInfo_ParamQDF")
    qdf.Parameters(0) = strModelGrid
    Set rcs = qdf.OpenRecordset
    'Make sure there are records
    If rcs.RecordCount <> 1 Then
        'raise an error
        Exit Function
       Else
        mRows = rcs("NumberOfRows")
        mColumns = rcs("NumberOfColumns")
        mLayers = rcs("NumberOfLayers")
```

```
        rcs.Close: Set rcs = Nothing
        qdf.Close: Set qdf = Nothing
    End If
Return

SUB_CheckGEM_WHT:
    Dim tmpDBName As String, tmpModelGrid As String
  myINISection.filename = strGEMFileOut
  myINISection.SectionName = "GEM_Wheat"

    'Call PrivIniRegister(sSectionName:="GEM_Wheat", sIniFileName:=strGEMFileOut)
    tmpDBName = myINISection.GetEntry(EntryName:="Database", DefaultValue:="")
    If tmpDBName = "" Then
         myINISection.SetEntry EntryName:="Database", EntryValue:=strDBName
        Else
         If tmpDBName <> strDBName Then
            Exit Function 'raise an error
         End If
    End If

    tmpModelGrid = myINISection.GetEntry(EntryName:="ModelGrid", DefaultValue:="")
    tmpModelGrid = Trim$(tmpModelGrid)
    strModelGrid = Trim$(strModelGrid)
    If tmpModelGrid = "" Then
         myINISection.SetEntry EntryName:="ModelGrid", EntryValue:=strModelGrid
        Else
         If tmpModelGrid <> strModelGrid Then
            Exit Function 'raise an error
         End If
    End If
Return


End Function
```

Kansas Geological Survey
Open-file Report

Disclaimer

The Kansas Geological Survey does not guarantee this document to be free from errors or inaccuracies and disclaims any responsibility or liability for interpretations based on data used in the production of this document or decisions based thereon. This report is intended to make results of research available at the earliest possible date, but is not intended to constitute final or formal publication.