SANDIA REPORT

SAND99-2470
Unlimited Release
Printed March 2000

Construction of File Database Management

Kyle Merrill

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: http://www.ntis.gov/ordering.htm

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND99-2470
Unlimited Release
Printed March 2000

Construction of File Database Management

Kyle Merrill
Materials Aging and Reliability Interfaces
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0340

Abstract

This work created a database for tracking data analysis files from multiple lab techniques and equipment stored on a central file server. Experimental details appropriate for each file type are pulled from the file header and stored in a searchable database. The database also stores specific location and self-directory structure for each data file. Queries can be run on the database according to file type, sample type or other experimental parameters. The database was constructed in Microsoft Access and Visual Basic was used for extraction of information from the file header.

Table of Contents

Figures and Tables

Title: Construction of File Database Management

Problem

There was a need for a database that would keep track of all the files from different lab experiments. This was important because the vast amount of data files would be able to be better managed. One sample would have multiple experiments performed on it and through performing a query the different data files associated with the specific sample could be found.

As the experiments were performed and the data recorded the files were saved on the server. Many experiments were performed with different kinds of software and equipment. This resulted in enormous amounts of data files being saved on the server throughout the years. The problem was trying to remember where the files were, when information was needed on a certain experiment sample. Also those that performed the experiments would like to be able to compare a sample that had different kinds of experiments performed on it. The Find File method, contained within windows, took huge amounts of time to look through the server for files. Also with this method the ability to narrow down the search criteria was limited.

One problem was the programming skills that were needed to accomplish this task were lacking. In counseling with the managers the database developer decided to learn Visual Basic 6.0. This was accomplished through buying some books and taking a one-week course taught by a Microsoft certified teacher. This gave the needed basic skills to start programming source code that would convert the file header into a format that could be imported into the database tables.

Solution

The header information is extracted from the data files and then recorded in the database. The information in the file header contains the parameters used during the experiment. This will help recall what data file is needed. The user can run a query in the database, searching for the different parameters that tell the user what data file they are looking for.

A program is written to convert the header format into a format that can be imported into the database. This involves different source code in the program, because each set of software and equipment has a different type of file header. The source code needs to read the header information from the data files and then write it to a temporary file in the correct format. The correct format consists of text all on one line separated by semicolons. Some of the files are in ASCII format and some are in Binary format. This requires different techniques of reading the file header from the data file.

This database needs to be easy to use so that one, it will be used, and two, it will save time in finding the needed file information. As this is accomplished, time will be saved and the information obtained from the experiments will be better utilized. The program extracts the needed file header information from the data files so it can be imported into the database. The database is user-friendly which means there is no data entry from the keyboard. This is extremely important, because in order for the database to get used, it has to be efficient and worthwhile to use.

Data Description

Microsoft Access was the database that was chosen for this project. It was chosen because it is fairly simple to use and is already on all the computers at Sandia. At the first of the project when trying to import the files directly into the database it was found that the files needed to be in an Access specific format. There were nine different file formats and none of them could be imported directly into Access. In discussing the solution, it was decided a program needed to be written to convert the file header information into a format that could be imported into Access. As Table 1 shows most of the file headers are in ASCII format which made it easy to read the information in the data file headers.

Table 1 Experiment types, perspective File Formats, and Module Filenames

Type of Experiment	Header Format	Data Format	Module Filename
Resistance	ASCII	ASCII	ModResistance.bas
PC Explorer	ASCII	Binary	ModPCExplorer.bas
Profilometer	ASCII	ASCII	ModProf.bas
Quadstar	ASCII	ASCII	ModQuadstar.bas
SEM	ASCII	Binary	ModSem.bas
AES	ASCII	Binary	ModAes.bas
Wyko	Binary	Binary	ModWyko.bas
Pin-on-Disk	ASCII	ASCII	ModPin.bas
Probe	ASCII	ASCII	ModProbe.bas

The header started at the beginning of the data file for the experiments that had an ASCII header format. This made it easy to read the file information since all that needed to happen was to start reading the file a line at a time, starting at the beginning. The files that had an ASCII header and Binary data were not a problem, because this project dealt with only importing the header information into the database.

The Wyko experiment created more of a problem because both the header and data were in Binary format. Also the header was at the end of the data file. The company that wrote the software was contacted and they provided detailed information about the file and its format. This included what kind of variables the file header contained and how many bytes were associated with each variable. To help know at what byte location the variable from the binary file needed to be read at, a Hex Editor was used, along with the information obtained from the company. The Hex Editor was freeware obtained from the Internet.

Software Development

There were many different tasks that needed to be accomplished through programming. One of the tasks was to be able to select the correct drive, folder, and file. The first form, (see Fig. 1), had command buttons that when selected source code would load the second form onto the screen. The second form, (see Fig. 2), looked like a file open dialog box. Also there was programming code that calls a specific sub procedure related to the command button selected on the first form. Each sub procedure reads the necessary information from the lab experiment file associated with the sub procedure and then writes the information to temporary files that were created.

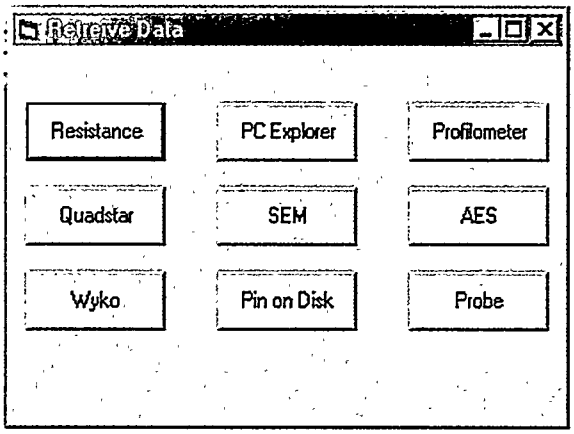

Fig. 1 First Form seen when Retrieve Data Macro is Run

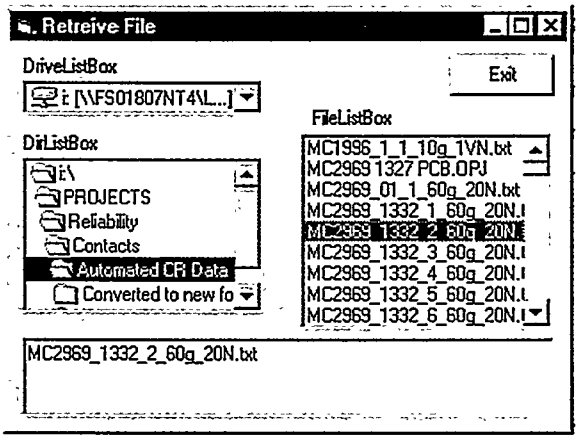

Fig. 2 Form seen when any of nine command buttons selected on Fig 1

The FormGroup.frm, (see Appendix A), was the first code to be executed when the program was run. The nine command buttons, (see Fig. 1), were created into a control array by naming them the same name in the property window of the design form. When this occured each button was assigned an index number starting with zero. The index numbers were used in a Select Case Function, which executes the code under the certain index number corresponding to the command button that was selected. FormGroup.frm also executed the FormGetFile.Show, which let the user see the form in Fig. 2. Also three temporary files were either created or the data in these files was erased when any one of the command buttons was selected.

Next in the program was the FromGetFile.frm code, (see Appendix A), which was the code from the form in Fig. 2. This code let the user navigate through the different drives, directories, folders, and files that were on the computer. Once a file was selected that was when the main part of the code executed. There was an "If Then ElseIf Function" that calls a certain module if the specific Boolean variable was true. This specific variable was set to true corresponding to the command button selected at the beginning. This code also set up the variables necessary to read and write the data from the data file to the temporary file

All of the modules accomplished the basic goal of reading the data from the correct file and writing the formatted information to the temporary files. Because the data files had differences in the header format there was the need to write code for each type of experiment. To see the actual code written in the modules, refer to Appendix B. In order to read the files that were in ASCII format, the program read the file a line at a time and then extracted the needed information from the line of text. The information was then written to the temporary files. One of the problems experienced was that the file headers were not of the same length.

ModAes.bas, ModPCExplorer.bas, and ModSem.bas all had extremely similar code because the file headers all were comparable. There were a few differences that required code to be written for each file. The differences were one file would be missing a line or have more lines in the header than the other file type. All of these files had the possibility of having multiple parameters in the header, therefore the code wrote those parameters to the third temporary file. ModSem.bas and ModAes.bas read in the header information differently, but the way the code wrote the information to the temporary file was adjusted so they both could be imported into the same tables in the database.

ModQuadstar.bas and ModResistance.bas were also similar to the previous modules in that they also wrote to three temporary files. The Quadstar data file was separated by tabs instead of spaces so the Split Function had to specify to Split the text that was separated by tabs. The Resistance data file had fourteen lines of text and also had the possibility of having multiple lines after them, depending on the file. The ModResistance.bas code read in the lines of text and then could tell whether or not there were multiple lines after the original fourteen.

The ModPin.bas, ModProbe.bas, and ModProf.bas modules all wrote to only two temporary files. One of the problems experienced by the Pin data was that some of the text lines contained a quotation mark. Access did not like importing this data, even though semicolons separated it, when it had a quotation mark in it. Code was written to replace the quotation mark with a blank space. The Probe data file had two lines of text, which did not have a space after the semicolon while all the other lines had one space. This required two lines of code to be written specifically dealing with those two lines.

In the ModWyko.bas module it was necessary to read the data file as a random access file. With knowledge obtained from the company that produced the Wyko software and a Hex Editor the data file could be read. Source code could be written to read a certain number of bytes for the specific type of variable, pointed at a specific location in the file. Then the binary variables were converted to ASCII a variable at a time, with CStr Function, and then were written to the temporary files.

Database Management

The program enabled the database user to input all of the data by using the mouse. This accomplished one of the objectives of making the database easy to use by having no keyboard data entry. The source code was created into an executable file, which was then used in a macro in Access by stating the path and filename of the executable. When the Retrieve Data macro ran it called the executable file which started the program. Once a command button on the first form was selected, (Fig.1) another form opens, (Fig. 2) which was similar to an open file form and had an exit button. The file or files to be input into the database could be found by going to the correct directory and folder and then clicking on the filename. The program then read the necessary information from the file and wrote it to the temporary files in the correct format. Now that the temporary files contained the file information in the correct format they could be imported into the Access database.

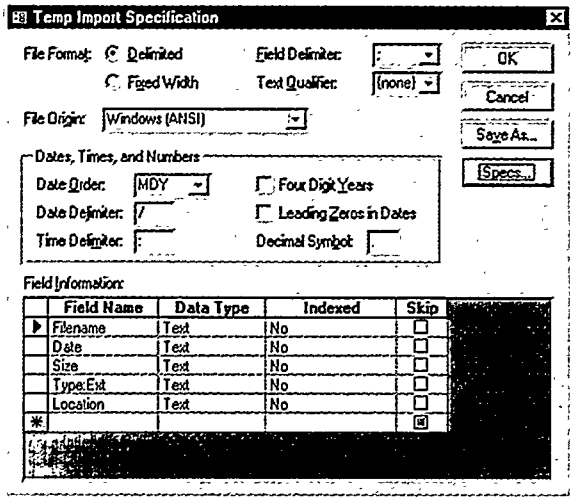

Fig. 3 MS Access Import Wizard's Advanced Specifications

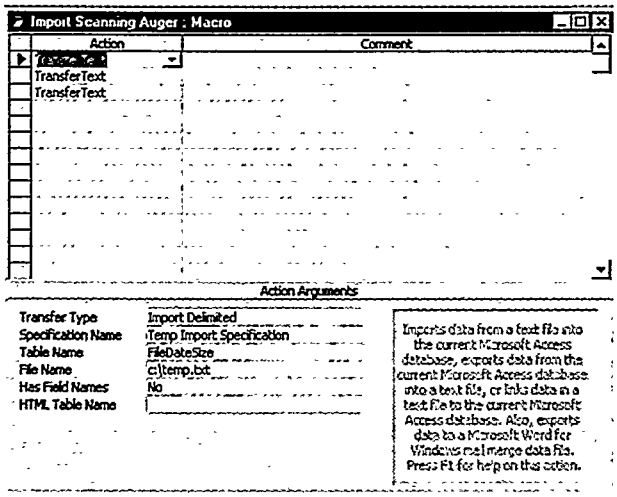

Fig. 4 Macro design time showing the Transfer Text Action and Arguments

Once all the files from a specific experiment had been selected the Exit button could be clicked to leave the program and macro. There were import macros for each different kind of file type and experiment. Indicating the Specification Name, Table Name, and File Name (See Fig. 4) was how the import macro was created. In order to produce the Specification Name the developer had to use Access' import features. Under the file menu was a Get External Data command, and then the Import option was selected. As the developer went through the different steps the computer asked for, there was an advanced button that when selected, the Field Names and Data Type could then be entered in to correspond to the matching table fields (See Fig. 3). This import specification was then saved and the name was placed in the transfer text macro settings corresponding to the correct temporary file and table (See Fig. 4). Under File Name the developer inputed the path and name of the specific temporary file being imported. This needed to correspond to the table entered in the Table Name category where the temporary file data was going to be imported.

FileDateSize : Table

Filename	Date	Size	Type:Ext	Location
980702 17c rotor, a hi.opd	6/30/99 8:48:46 AM	449176	opd	C:\documents\980702 17c rotor, a hi.opd
990403_10.sem	6/8/99 11:30:51 AM	104949	sem	A:\990403_10.sem
990405_3.spe	6/8/99 11:30:49 AM	14109	spe	A:\990405_3.spe
990407f.dat	6/30/99 10:03:04 AM	214137	dat	I:\USERS\kyle\files for Kyle\990407f.dat
a&b.txt	6/30/99 10:07:42 AM	544	txt	I:\USERS\kyle\files for Kyle\a&b.txt
D9122.SPE	4/12/99 11:46:34 AM	17970	SPE	I:\DATA\XPS\Apr99\D9122.SPE
FILEFRM2.ASC	7/7/99 9:05:47 AM	2894	ASC	I:\USERS\kyle\FILEFRM2.ASC

Fig. 5 FileDateSize Table from the HeaderDataStorage Database

When the specific import macro ran, the information in the temporary files were written to the corresponding tables in the database, resulting in something similar to Fig 5. This particular table contained every data file regardless of its type. The main advantage of this table was the Location field, which helped the user locate a specific file on the server or wherever the file was located. The other tables contained Fields equivalent to the specific experiments data file header. Some of the experiments had one table and some had two, depending on what the data was like in the file header.

Conclusion

The time to input the data into the database is insignificant compared to the time that will be saved in finding the experiment information that is needed. One can learn how to input the data into the database in minutes. The process is extremely easy and as stated previously requires no keyboard entry.

This project will better utilize the experiments performed on the samples in many ways. One sample will be tested and go through different experiments. By doing a query in the database, the different files associated with the specific sample or with related samples can be found. Large sets of data will be managed and the data can be cross-referenced from multiple analysis. This will result in time saved in trying to find the needed information on the samples tested and will better utilize all the experiments performed.

Future Modifications

The Visual Basic program is set up in a way that if a new experiment type needs to be added to the database, this can be accomplished with little change to the program. One experiment type is not included in the database because the data file is in a binary format and sufficient information couldn't be obtained from the company that produced the software.

The Quadstar data files also are in binary format and the company does not supply the needed information to read the files. This software has the ability to convert the data and header information into an ASCII file. The ASCII data file can then be read and put into the correct format to be imported into the database. If the knowledge to read these files is obtained through the information that is supplied from these companies then these files in their binary format can also be imported into the database.

Appendix A

FormGroup.frm

```
' This is the first form that you see. When you click on one of the command buttons
' it opens the Get File form and also assigns the specific boolean variable to true which is
' used in the if then statements
Option Explicit

' Declares variables as Public which means they are shared from form to form

Public blnResistance As Boolean, blnAes As Boolean
Public blnProf As Boolean
Public blnSem As Boolean, blnWyko As Boolean
Public blnPCExplorer As Boolean
Public blnProbe As Boolean, blnPin As Boolean
Public blnQuadstar As Boolean
Public mfso As New FileSystemObject        ' In using this type of variable one must go to
                                           ' Project menu and then References and then
                                           ' check Microsoft Scripting Runtime
Private Sub cmdMain_Click(Index As Integer)
   Call EraseTempFiles                     ' Creates temp files or erases data in them
   frmGetFile.Show

   Select Case Index                       ' A control array which sets the boolean value
      Case 0                               ' to true depending on which button is selected
         blnResistance = True
      Case 1
         blnPCExplorer = True
      Case 2
         blnProf = True
      Case 3
         blnQuadstar = True
      Case 4
         blnSem = True
      Case 5
         blnAes = True
      Case 6
         blnWyko = True
      Case 7
         blnPin = True
      Case 8
         blnProbe = True
   End Select
End Sub

' Creates three temporary files to write the converted header into
Public Sub EraseTempFiles()
   Call mfso.CreateTextFile("c:\temp.txt", True)
   Call mfso.CreateTextFile("c:\temp2.txt", True)
   Call mfso.CreateTextFile("c:\temp3.txt", True)
End Sub
```

FormGetFile.frm

```
' This form lets you select the specific file that you want from a file open type form
Option Explicit

' This correlates to Exit command button and ends the program when clicked
Private Sub cmdExit_Click()
   End
End Sub

' These next two Sub Procedures makes it so you can select the specific file you want
' from the correct directory and folder

Private Sub dirDirBox_Change()
  ' Update the file path to the directory path
  filFileBox.Path = dirDirBox.Path
End Sub

Private Sub drvDriveBox_Change()

  On Error GoTo errorhandler

    ' Update the directory path to the drive
    dirDirBox.Path = drvDriveBox.Drive
    Exit Sub

errorhandler:
    Dim message As String

    ' Check for device unavailable error
    If Err.Number = 68 Then
      Dim r As Integer
      message = "Drive is not available."
      r = MsgBox(message, vbRetryCancel + vbCritical, _
            "VBHTP: Chapter 14")

      ' Determine where control should resume
      If r = vbRetry Then
        Resume
      Else   ' Cancel was pressed.
        Exit Sub
      End If

    Else
      Call MsgBox(Err.Description, vbOKOnly + vbExclamation)
      Exit Sub
    End If
End Sub

Public Sub filFileBox_Click
   Dim theFile As File
   Dim mFile As File, mFile2 As File, mFile3 As File
```

```
Dim mTxtStream As TextStream
   Dim mts As TextStream, mts2 As TextStream, mts3 As TextStream

' This code assigns the variable of type file to the 3 temp files
' Then assigns the variable of type TextStream to the file for Appending

   Set mFile = FormMain.mfso.GetFile("c:\temp.txt")
   Set mts = mFile.OpenAsTextStream(ForAppending)
   Set mFile2 = FormMain.mfso.GetFile("c:\temp2.txt")
   Set mts2 = mFile2.OpenAsTextStream(ForAppending)
   Set mFile3 = FormMain.mfso.GetFile("c:\temp3.txt")
   Set mts3 = mFile3.OpenAsTextStream(ForAppending)

' Assigns the variable of type file to the file that is clicked with mouse
' Then assigns the TextStream variable to Open for Reading from clicked file

   Set theFile = FormMain.mfso.GetFile(filFileBox.Path & "\" & _
         filFileBox.List(filFileBox.ListIndex))
   Set mTxtStream = theFile.OpenAsTextStream(ForReading)

' This statement writes the information in the parenthesis to the file temp.txt
   mts.WriteLine (theFile.Name & ";" & _
            theFile.DateCreated & ";" & _
            theFile.Size & ";" & _
            Right$(theFile.Name, 3) & ";" & _
            theFile.Path)
   mts.Close
   txtDisplay.Text = theFile.Name

' These if statements call the correct procedure depending on
' what command button was clicked in the very first form

   If FormMain.blnAes = True Then
      Call AesData(mTxtStream, mts2, mts3, theFile)
   ElseIf FormMain.blnResistance = True Then
      Call ResistanceData(mTxtStream, mts2, mts3, theFile)
   ElseIf FormMain.blnPCExplorer = True Then
      Call PCExplorerData(mTxtStream, mts2, mts3, theFile)
   ElseIf FormMain.blnProf = True Then
      Call ProfData(mTxtStream, mts2, mts3, theFile)
   ElseIf FormMain.blnSem = True Then
      Call SemData(mTxtStream, mts2, mts3, theFile)
   ElseIf FormMain.blnWyko = True Then
      Call WykoData(mts2, theFile)
   ElseIf FormMain.blnProbe = True Then
      Call ProbeData(mTxtStream, mts2, theFile)
   ElseIf FormMain.blnPin = True Then
      Call PinData(mTxtStream, mts2, theFile)
   ElseIf FormMain.blnQuadstar = True Then
      Call QuadstarData(mTxtStream, mts2, mts3, theFile)
   End If
End Sub
```

Appendix B

ModAes.bas

```
' Variables in parenthesis are passed to and from the get file form

Public Sub AesData(mTxtStream As TextStream, mts2 As TextStream, _
        mts3 As TextStream, theFile As File)
  Dim c As String
  Dim aa(60) As String, bb() As String
  Dim z As Integer, blankposition As Integer
  Dim j As Integer, i As Integer
  Dim mark(60) As Integer

  On Error GoTo ErrorhandlerAes

' Loops until EOFH is read which is at the end of the Header, blankposition makes
' certain that the array aa(z) has no blank lines

  z = 1
  Do Until aa(z) = "EOFH"
    blankposition = InStr(1, aa(z), " ")
    If blankposition = 0 Then
      z = z - 1
    End If
    z = z + 1
    aa(z) = mTxtStream.ReadLine
    mark(z) = InStr(1, aa(z), ":") + 2
    If Left$(aa(z), 15) = "SpectralRegDef:" Then
      i = z
    End If          ' If there is just one line starting with SpectralRegDef
  Loop              ' then i will equal 25 so the j loop will just loop once

' This loop writes the parameters that may have more than one line to the 3rd temp file

  For j = 25 To i                       ' line 25 is where the multiple parameters
    c = Trim$(Mid$(aa(j), mark(j)))     ' in the header start
    bb = Split(c)
    mts3.WriteLine (theFile.Name & ";" & _
      bb(1) & ";" & bb(2) & ";" & bb(4) & ";" & _
      bb(5) & ";" & bb(6) & ";" & bb(7) & ";" & _
      bb(10) & ";" & bb(11))
  Next j
  mts3.Close

' Trim Function trims any spaces around text stream and Mid Function writes
' the text stream of aa() array starting at position mark()

  mts2.WriteLine (theFile.Name & ";" & _
    Trim(Mid(aa(1), mark(1))) & ";" & Trim(Mid(aa(2), mark(2))) & ";" & _
    Trim(Mid(aa(3), mark(3))) & ";" & Trim(Mid(aa(4), mark(4))) & ";" & _
    Trim(Mid(aa(5), mark(5))) & ";" & "" & ";" & _
```

```
    Trim(Mid(aa(8), mark(8))) & ";" & _
    Trim(Mid(aa(9), mark(9))) & ";" & Trim(Mid(aa(10), mark(10))) & ";" & _
    Trim(Mid(aa(11), mark(11))) & ";" & Trim(Mid(aa(12), mark(12))) & ";" & _
    Trim(Mid(aa(13), mark(13))) & ";" & Trim(Mid(aa(14), mark(14))) & ";" & _
    Trim(Mid(aa(15), mark(15))) & ";" & Trim(Mid(aa(16), mark(16))) & ";" & _
    Trim(Mid(aa(17), mark(17))) & ";" & Trim(Mid(aa(18), mark(18))) & ";" & _
    Trim(Mid(aa(19), mark(19))) & ";" & Trim(Mid(aa(20), mark(20))) & ";" & _
    Trim(Mid(aa(21), mark(21))) & ";" & Trim(Mid(aa(22), mark(22))) & ";" & _
    Trim(Mid(aa(23), mark(23))) & ";" & Trim(Mid(aa(24), mark(24))) & ";" & _
    "" & ";" & _
    Trim(Mid(aa(z - 4), mark(z - 4))) & ";" & _
    Trim(Mid(aa(z - 3), mark(z - 3))) & ";" & _
    Trim(Mid(aa(z - 2), mark(z - 2))) & ";" & _
    Trim(Mid(aa(z - 1), mark(z - 1))))              ' the aa(z-1to4) writes the four lines
  mts2.Close                                        ' after the line that may have
                                                    ' multiple parameters

Exit Sub        ' The blank from the "" five lines up makes this module compatible
                ' with the Sem module for input into the database
ErrorhandlerAes:
  MsgBox "This file cannot be processed please try again.", _
    vbOKOnly, "Bad File"
End Sub
```

ModPCExplorer.bas

```
' Variables in parenthesis are passed to and from frmGetFile code and this module

Public Sub PCExplorerData(mTxtStream As TextStream, mts2 As TextStream, _
        mts3 As TextStream, theFile As File)
   Dim c As String
   Dim aa(60) As String, bb() As String
   Dim z As Integer, blankposition As Integer, i As Integer
   Dim j As Integer
   Dim mark(60) As Integer
   Dim markz3 As Integer, markz2 As Integer, markz1 As Integer

' If an error is experienced at anytime, then this line of code executes
   On Error GoTo ErrorhandlerPCExplorer

' Loops until text EOFH is read which is at the end of file header
   z = 1
   Do Until aa(z) = "EOFH"
      blankposition = InStr(1, aa(z), " ")          ' If there are any  blank lines z will
      If blankposition = 0 Then                     ' not increment
         z = z - 1
      End If
      z = z + 1
      aa(z) = mTxtStream.ReadLine                   ' Reads one line of text
      mark(z) = InStr(1, aa(z), ":") + 2            ' Marks position 2 places after :
      If Left$(aa(z), 15) = "SpectralRegDef:" Then
         i = z                                      ' Multiple lines may start with this text
      End If
   Loop                              ' i records last value of z which starts with this text

   markz3 = InStr(1, aa(z - 3), ":") + 2
   markz2 = InStr(1, aa(z - 2), ":") + 2
   markz1 = InStr(1, aa(z - 1), ":") + 2

   mts2.WriteLine (theFile.Name & ";" & _
            Trim(Mid(aa(2), mark(2))) & ";" & _
            Trim(Mid(aa(3), mark(3))) & ";" & _
            Trim(Mid(aa(4), mark(4))) & ";" & _
            Trim(Mid(aa(5), mark(5))) & ";" & _
            Trim(Mid(aa(9), mark(9))) & ";" & _
            Trim(Mid(aa(11), mark(11))) & ";" & _
            Trim(Mid(aa(20), mark(20))) & ";" & _
            Trim(Mid(aa(21), mark(21))) & ";" & _
            Trim(Mid(aa(25), mark(25))) & ";" & _
            Trim(Mid(aa(26), mark(26))) & ";" & _
            Trim(Mid(aa(27), mark(27))) & ";" & _
            Trim(Mid(aa(28), mark(28))) & ";" & _
            Trim(Mid(aa(30), mark(30))) & ";" & _
            Trim(Mid(aa(33), mark(33))) & ";" & _
            Trim(Mid(aa(34), mark(34))) & ";" & _
            Trim(Mid(aa(35), mark(35))) & ";" & _
```

```
            Trim(Mid(aa(36), mark(36))) & ";" & _
            Trim(Mid(aa(37), mark(37))) & ";" & _
            Trim(Mid(aa(38), mark(38))) & ";" & _
            Trim(Mid(aa(39), mark(39))) & ";" & _
            Trim(Mid(aa(z - 3), markz3)) & ";" & _
            Trim(Mid(aa(z - 2), markz2)) & ";" & _
            Trim(Mid(aa(z - 1), markz1)))

    mts2.Close                              ' Trims any blank spaces around the text
                                            ' and Mid starts text aa() at position mark()
    For j = 40 To i
      c = Trim$(Mid$(aa(j), mark(40)))
      bb = Split(c)
      mts3.WriteLine (theFile.Name & ";" & _
            Trim$(Mid$(aa(4), mark(4))) & ";" & _
            bb(1) & ";" & bb(2) & ";" & _
            bb(4) & ";" & bb(5) & ";" & _
            bb(6) & ";" & bb(7) & ";" & _
            bb(10) & ";" & bb(11))
    Next j
    mts3.Close
    Exit Sub        ' Exits Sub since no error occured so error message does not appear

ErrorhandlerPCExplorer:
    MsgBox "This file cannot be processed please try again.", _
      vbOKOnly, "Bad File"
End Sub
```

ModPin.bas

```
Public Sub PinData(mTxtStream As TextStream, mts2 As TextStream, theFile As File)
   Dim aa(60) As String
   Dim z As Integer, blankposition As Integer
   Dim mark(60) As Integer
   Dim position As Integer
   Dim Pointer As Integer
   Dim Quote As String

   On Error GoTo Errorhandler2
   z = 1
   Do Until Left(aa(z), 10) = "Processing"
      blankposition = InStr(1, aa(z), " ")
      If blankposition = 0 Then
         z = z - 1
      End If
      z = z + 1
      aa(z) = mTxtStream.ReadLine
      mark(z) = InStr(1, aa(z), ":") + 2

      For position = 1 To Len(aa(z))          ' Access had trouble importing text
         Quote = Mid(aa(z), position, 1)      ' because it had a quote " in the line
         Pointer = InStr(position, aa(z), """")  ' this code gets ride of the quote and
         If Quote = """" Then                 ' replaces it with a blank space
            Mid(aa(z), Pointer, 2) = " "
         End If                          ' Mid(Source text, Starting character position,
      Next position                      ' # of characters to select)
   Loop
                                         ' InStr(Starting search position, string to search,
   mark(2) = InStr(1, aa(2), ",") + 2    ' what to search for) this returns position in string
                                         ' of what searching for
   mts2.WriteLine (theFile.Name & ";" & _
         Trim(Mid(aa(2), mark(2))) & ";" & _
         Trim(Mid(aa(3), mark(3))) & ";" & _
         Trim(Mid(aa(4), mark(4))) & ";" & _
         Trim(Mid(aa(5), mark(5))) & ";" & _
         Trim(Mid(aa(6), mark(6))) & ";" & _
         Trim(Mid(aa(7), mark(7))) & ";" & _
         Trim(Mid(aa(8), mark(8))) & ";" & _
         Trim(Mid(aa(9), mark(9))) & ";" & _
         Trim(Mid(aa(10), mark(10))) & ";" & _
         Trim(Mid(aa(13), mark(13))) & ";" & _
         Trim(Mid(aa(14), mark(14))) & ";" & _
         Trim(Mid(aa(15), mark(15))) & ";" & _
         Trim(Mid(aa(16), mark(16))))
   mts2.Close
Exit Sub
Errorhandler2:
   MsgBox "This file cannot be processed please try again.", vbOKOnly, "Bad File"
End Sub
```

ModProbe.bas

```
Public Sub ProbeData(mTxtStream As TextStream, _
        mts2 As TextStream, theFile As File)

  Dim aa(20) As String
  Dim z As Integer
  Dim mark(20) As Integer

  On Error GoTo ErrorhandlerProbe

  For z = 1 To 14                          ' Reads 14 lines
    aa(z) = mTxtStream.ReadLine
    mark(z) = InStr(1, aa(z), ":") + 2
  Next z

  mark(10) = InStr(1, aa(10), ":") + 1     ' No space after the semicolon in data file is
  mark(13) = InStr(1, aa(13), ":") + 1     ' why these lines are needed

' Writes text pulled from theFile onto one line separated by semicolons and puts it into a
' temp file

  mts2.WriteLine (theFile.Name & ";" & _
      Trim$(Mid$(aa(3), mark(3))) & ";" & _
      Trim$(Mid$(aa(4), mark(4))) & ";" & _
      Trim$(Mid$(aa(5), mark(5))) & ";" & _
      Trim$(Mid$(aa(6), mark(6))) & ";" & _
      Trim$(Mid$(aa(7), mark(7))) & ";" & _
      Trim$(Mid$(aa(8), mark(8))) & ";" & _
      Trim$(Mid$(aa(9), mark(9))) & ";" & _
      Trim$(Mid$(aa(10), mark(10))) & ";" & _
      Trim$(Mid$(aa(11), mark(11))) & ";" & _
      Trim$(Mid$(aa(12), mark(12))) & ";" & _
      Trim$(Mid$(aa(13), mark(13))) & ";" & _
      Trim$(Mid$(aa(14), mark(14))))
  mts2.Close

Exit Sub
ErrorhandlerProbe:
  MsgBox "This file cannot be processed please try again.", _
    vbOKOnly, "Bad File"
End Sub
```

ModProf.bas

```
' Variables are passed from the get file form to be used in this module
' and then they are passed back

Public Sub ProfData(mTxtStream As TextStream, mts2 As TextStream, _
        mts3 As TextStream, theFile As File)
   Dim aa(50) As String, bb(50) As String
   Dim mark(50) As Integer
   Dim z As Integer, blankposition As Integer

   On Error GoTo ErrorhandlerProf

   For z = 1 To 45                          ' Loops 45 times reads text into array aa()
      aa(z) = mTxtStream.ReadLine           ' and marks where colon occurs, then array
      mark(z) = InStr(1, aa(z), ":") + 2    ' bb() contains text starting at position mark()
      bb(z) = Trim(Mid(aa(z), mark(z)))
  Next z

   mts2.WriteLine (theFile.Name & ";" & _
            bb(1) & ";" & bb(2) & ";" & _
            bb(3) & ";" & bb(4) & ";" & _
            bb(5) & ";" & bb(6) & ";" & _
            bb(7) & ";" & bb(8) & ";" & _
            bb(9) & ";" & bb(10) & ";" & _
            bb(11) & ";" & bb(12) & ";" & _
            bb(13) & ";" & bb(14) & ";" & _
            bb(15) & ";" & bb(16) & ";" & _
            bb(17) & ";" & bb(18) & ";" & _
            bb(19) & ";" & bb(20) & ";" & _
            bb(21) & ";" & bb(22) & ";" & _
            bb(23) & ";" & bb(24) & ";" & _
            bb(25) & ";" & bb(26) & ";" & _
            bb(27) & ";" & bb(28) & ";" & _
            bb(29) & ";" & bb(30) & ";" & _
            bb(31) & ";" & bb(32) & ";" & _
            bb(33) & ";" & bb(34) & ";" & _
            bb(35) & ";" & bb(36) & ";" & _
            bb(37) & ";" & bb(38) & ";" & _
            bb(39) & ";" & bb(40) & ";" & _
            bb(41) & ";" & bb(42) & ";" & _
            bb(43) & ";" & bb(44))
   mts2.Close

   Exit Sub
ErrorhandlerProf:
   MsgBox "This file cannot be processed please try again.", _
      vbOKOnly, "Bad File"
End Sub
```

ModQuadstar.bas

```
Public Sub QuadstarData(mTxtStream As TextStream, _
        mts2 As TextStream, mts3 As TextStream, _
        theFile As File)

  Dim j As Integer
  Dim aa As String, bb() As String
  Dim cc1 As String, cc3 As String, cc5 As String

  On Error GoTo ErrorhandlerQuadstar

  aa = mTxtStream.ReadLine              ' Data File contains all the text on one line

  bb = Split(aa, vbTab)             ' Splits text seperated by a tab assigns to bb()
  For j = 25 To 101 Step 6
    cc1 = bb(j)                     ' Assigns to cc1,3,and 5 every other value
    cc3 = bb(j + 2)                 ' in that particular text line
    cc5 = bb(j + 4)
    If cc1 = "Date" Then
      GoTo Continue
    Else
      mts3.WriteLine (theFile.Name & ";" & _
          cc1 & ";" & cc3 & ";" & cc5)
    End If
  Next j
  mts3.Close

Continue:           ' Trim Function trims line of text starting at position Length of text
                    ' minus length of standard text at beginning of text line

  mts2.WriteLine (theFile.Name & ";" & bb(0) & ";" & _
          Trim(Mid(bb(3), 1, Len(bb(3)) - 7)) & ";" & _
          bb(4) & ";" & Trim(Mid(bb(9), 1, Len(bb(9)) - 25)) _
          & ";" & Trim(Mid(bb(12), 1, Len(bb(12)) - 20)) _
          & ";" & Trim(Mid(bb(15), 1, Len(bb(15)) - 18)) _
          & ";" & Trim(Mid(bb(18), 1, Len(bb(18)) - 28)) _
          & ";" & Trim(Mid(bb(21), 1, Len(bb(21)) - 13)))

  mts2.Close

  Exit Sub

ErrorhandlerQuadstar:
  MsgBox "This file cannot be processed please try again.", _
    vbOKOnly, "Bad File"

End Sub
```

ModResistance.bas

```
Public Sub ResistanceData(mTxtStream As TextStream, mts2 As TextStream, _
        mts3 As TextStream, theFile As File)

  Dim aa(40) As String, bb() As String
  Dim c As String
  Dim mark(40) As Integer
  Dim blankposition As Integer
  Dim z As Integer, i As Integer, j As Integer
  Dim Answer As Integer

  On Error GoTo Errorhandler2

  For z = 1 To 14                          ' Reads a line at a time from one to fourteen
    aa(z) = mTxtStream.ReadLine            ' and also marks the position 2 places after :
    mark(z) = InStr(1, aa(z), ":") + 2
  Next z

  Do Until Left(aa(z), 4) = "Load"
    blankposition = InStr(1, aa(z), "")
    If blankposition = 0 Then              ' If there is a blank line then z doesn't increment
      z = z - 1
    End If
    z = z + 1
    aa(z) = mTxtStream.ReadLine
    mark(z) = InStr(1, aa(z), ":") + 2
    If z = 16 And aa(z) = "" Then          ' Data file contains no data so skips the
      GoTo Continue                        ' j For Next Loop
    End If
    i = z - 1
  Loop

  For j = 16 To i                ' Splits line of text separated by tab
    c = aa(j)                    ' and puts text in array bb()
    bb = Split(c, vbTab)
    mts3.WriteLine (theFile.Name & ";" & _
            bb(0) & ";" & bb(1) & ";" & _
            bb(2) & ";" & bb(3))
  Next j
  mts3.Close

Continue:                                  ' Trims text at marked position
  mts2.WriteLine (theFile.Name & ";" & _
        Trim(Mid(aa(1), mark(1))) & ";" & _
        Trim(Mid(aa(2), mark(2))) & ";" & _
        Trim(Mid(aa(3), mark(3))) & ";" & _
        Trim(Mid(aa(4), mark(4))) & ";" & _
        Trim(Mid(aa(5), mark(5))) & ";" & _
        Trim(Mid(aa(6), mark(6))) & ";" & _
        Trim(Mid(aa(7), mark(7))) & ";" & _
        Trim(Mid(aa(8), mark(8))) & ";" & _
```

```
            Trim(Mid(aa(9), mark(9))) & ";" & _
            Trim(Mid(aa(10), mark(10))) & ";" & _
            Trim(Mid(aa(11), mark(11))) & ";" & _
            Trim(Mid(aa(12), mark(12))) & ";" & _
            Trim(Mid(aa(13), mark(13))) & ";" & _
            Trim(Mid(aa(14), mark(14))))

    mts2.Close

    Exit Sub                    ' If data file contains header only and no data
Errorhandler2:                  ' Then error message 62 occurs
    If Err.Number = 62 Then
        Answer = MsgBox("This file contains no data." & vbCrLf & _
            "Do you want to save in Database.", vbYesNo, "Bad File")
        If Answer = vbYes Then
            Resume Next
        Else
            Exit Sub
        End If
    Else
        MsgBox "This file cannot be processed please try again.", _
            vbOKOnly, "Bad File"
    End If
End Sub
```

ModSem.bas

```
' For comments on code see Aes Module

Public Sub SemData(mTxtStream As TextStream, mts2 As TextStream, _
        mts3 As TextStream, theFile As File)

  Dim aa(60) As String, bb() As String, c As String
  Dim z As Integer, blankposition As Integer, i As Integer
  Dim j As Integer
  Dim mark(60) As Integer

  On Error GoTo Errorhandler2

  z = 1
  Do Until aa(z) = "EOFH"
    blankposition = InStr(1, aa(z), " ")
    If blankposition = 0 Then
      z = z - 1
    End If
    z = z + 1
    aa(z) = mTxtStream.ReadLine
    mark(z) = InStr(1, aa(z), ":") + 2
    If Left$(aa(z), 15) = "SpectralRegDef:" Then
      i = z
    End If
  Loop

  For j = 26 To i
  c = Trim$(Mid$(aa(j), mark(j)))
  bb = Split(c)
  mts3.WriteLine (theFile.Name & ";" & _
        bb(1) & ";" & bb(2) & ";" & _
        bb(4) & ";" & bb(5) & ";" & _
        bb(6) & ";" & bb(7) & ";" & _
        bb(10) & ";" & bb(11))
  Next j
  mts3.Close

  mts2.WriteLine (theFile.Name & ";" & _
    Trim(Mid(aa(1), mark(1))) & ";" & Trim(Mid(aa(2), mark(2))) & ";" & _
    Trim(Mid(aa(3), mark(3))) & ";" & Trim(Mid(aa(4), mark(4))) & ";" & _
    Trim(Mid(aa(5), mark(5))) & ";" & Trim(Mid(aa(8), mark(8))) & ";" & _
    Trim(Mid(aa(9), mark(9))) & ";" & Trim(Mid(aa(10), mark(10))) & ";" & _
    Trim(Mid(aa(11), mark(11))) & ";" & Trim(Mid(aa(12), mark(12))) & ";" & _
    Trim(Mid(aa(13), mark(13))) & ";" & Trim(Mid(aa(14), mark(14))) & ";" & _
    Trim(Mid(aa(15), mark(15))) & ";" & Trim(Mid(aa(16), mark(16))) & ";" & _
    Trim(Mid(aa(17), mark(17))) & ";" & Trim(Mid(aa(18), mark(18))) & ";" & _
    Trim(Mid(aa(19), mark(19))) & ";" & Trim(Mid(aa(20), mark(20))) & ";" & _
    Trim(Mid(aa(21), mark(21))) & ";" & Trim(Mid(aa(22), mark(22))) & ";" & _
    Trim(Mid(aa(23), mark(23))) & ";" & Trim(Mid(aa(24), mark(24))) & ";" & _
    Trim(Mid(aa(25), mark(25))) & ";" & _
```

```
        Trim(Mid(aa(z - 5), mark(z - 5))) & ";" & _
        Trim(Mid(aa(z - 4), mark(z - 4))) & ";" & _
        Trim(Mid(aa(z - 3), mark(z - 3))) & ";" & _
        Trim(Mid(aa(z - 2), mark(z - 2))) & ";" & _
        Trim(Mid(aa(z - 1), mark(z - 1))))
    mts2.Close
Exit Sub
Errorhandler2:
    MsgBox "This file cannot be processed please try again.", _
        vbOKOnly, "Bad File"
End Sub
```

ModWyko.bas

```
Public Sub WykoData(mts2 As TextStream, theFile As File)

    Dim strTime As String * 9          ' All string variables need to
    Dim strDate As String * 9          ' be assigned a specific number of bytes
    Dim OffsetDac As Single
    Dim SpareADC As Single             ' Single variables 4 bytes
    Dim MeasMode As String * 10        ' Single precision floating-point
    Dim Vibration As Single
    Dim Pixel_size As Single
    Dim MeasSlope As Single
    Dim Magnification As Single
    Dim Wavelength As Single
    Dim GlobalOriginX As Single, GlobalOriginY As Single
    Dim StageX As Single, StageY As Single
    Dim Pupil As Single
    Dim TiltX As Single, TiltY As Single
    Dim Use_XYR_cent As Integer        ' Integer variables 2 bytes
    Dim Use_XYR_spac As Integer
    Dim Pupil_diam As Single
    Dim XYR_x_spac As Single
    Dim Terms_String As String * 40
    Dim Data_Restore As String * 10
    Dim Data_Invert As String * 10
    Dim Filt_Type As String * 40
    Dim Vol_opt_String As String * 40
    Dim UseApodization As Integer
    Dim Aspect As Single
    Dim Title As String * 20
    Dim Note As String * 60

    On Error GoTo ErrorhandlerWyko

    Open theFile For Binary Access Read As #1      ' Reads specific number of bytes
    Get #1, 447614, strTime                        ' at position indicated, number of
    Open theFile For Binary Access Read As #2      ' bytes corresponds to the variable
    Get #2, 447624, strDate
    Open theFile For Binary Access Read As #3
    Get #3, 447633, OffsetDac
    Open theFile For Binary Access Read As #4
    Get #4, 447637, SpareADC
    Open theFile For Binary Access Read As #5
    Get #5, 447641, MeasMode
    Open theFile For Binary Access Read As #6
    Get #6, 447651, Vibration
    Open theFile For Binary Access Read As #7
    Get #7, 447655, Pixel_size
    Open theFile For Binary Access Read As #8
    Get #8, 447659, MeasSlope
    Open theFile For Binary Access Read As #9
    Get #9, 447663, Wavelength
```

```
Open theFile For Binary Access Read As #10
Get #10, 447667, Magnification
Open theFile For Binary Access Read As #11
Get #11, 447671, GlobalOriginX
Open theFile For Binary Access Read As #12
Get #12, 447675, GlobalOriginY
Open theFile For Binary Access Read As #13
Get #13, 447679, StageX
Open theFile For Binary Access Read As #14
Get #14, 447683, StageY
Open theFile For Binary Access Read As #15
Get #15, 448887, Pupil
Open theFile For Binary Access Read As #16
Get #16, 448891, TiltX
Open theFile For Binary Access Read As #17
Get #17, 448895, TiltY
Open theFile For Binary Access Read As #18
Get #18, 448899, Use_XYR_cent
Open theFile For Binary Access Read As #19
Get #19, 448901, Use_XYR_spac
Open theFile For Binary Access Read As #20
Get #20, 448903, Pupil_diam
Open theFile For Binary Access Read As #21
Get #21, 448907, XYR_x_spac
Open theFile For Binary Access Read As #22
Get #22, 448911, Terms_String
Open theFile For Binary Access Read As #23
Get #23, 448951, Data_Restore
Open theFile For Binary Access Read As #24
Get #24, 448961, Data_Invert
Open theFile For Binary Access Read As #25
Get #25, 448971, Filt_Type
Open theFile For Binary Access Read As #26
Get #26, 449011, Vol_opt_String
Open theFile For Binary Access Read As #27
Get #27, 449051, UseApodization
Open theFile For Binary Access Read As #31
Get #31, 449053, MaskingString
Open theFile For Binary Access Read As #28
Get #28, 449093, Aspect
Open theFile For Binary Access Read As #29
Get #29, 449097, Title
Open theFile For Binary Access Read As #30
Get #30, 449117, Note

' CStr converts the variable in parenthesis to String variable

mts2.WriteLine (theFile.Name & ";" & Trim(strDate) & ";" & _
    Trim(Title) & ";" & _
    Trim(Note) & ";" & CStr(OffsetDac) & ";" & _
    CStr(SpareADC) & ";" & Trim(MeasMode) & ";" & _
    CStr(Vibration) & ";" & CStr(Pixel_size) & _
```

```
        ";" & CStr(MeasSlope) & ";" & _
        CStr(Wavelength) & ";" & CStr(Magnification) & ";" & _
        CStr(GlobalOriginX) & ";" & CStr(GlobalOriginY) & ";" & _
        CStr(StageX) & ";" & CStr(StageY) & ";" & _
        CStr(Pupil) & ";" & CStr(TiltX) & ";" & _
        CStr(TiltY) & ";" & CStr(Use_XYR_cent) & ";" & _
        CStr(Use_XYR_spac) & ";" & CStr(Pupil_diam) & ";" & _
        CStr(XYR_x_spac) & ";" & Trim(Terms_String) & ";" & _
        Trim(Data_Restore) & ";" & Trim(Data_Invert) & ";" & _
        Trim(Filt_Type) & ";" & _
        Trim(Vol_opt_String) & ";" & _
        CStr(UseApodization) & ";" & _
        Trim(MaskingString) & ";" & _
        CStr(Aspect))

    mts2.Close

    Exit Sub

ErrorhandlerWyko:
    MsgBox "This file cannot be processed please try again.", _
        vbOKOnly, "Bad File"

End Sub
```

Distribution:

1	MS 0340	Wendy Cieslak, 1832
1	MS 0340	Michael Dugger, 1832
1	MS 0340	William McNamara, 1832
1	MS 0340	James Ohlhausen, 1832
1	MS 0340	Diane Peebles, 1832
1	MS 0340	Greg Poulter, 1832
1	MS 0340	Elizabeth Sorroche, 1832
1	MS 9018	Central Technical Files, 8940-2
2	MS 0899	Technical Library, 4916
1	MS 0612	Review & Approval Desk, 4912 For DOE/OSTI