Random Access Files on the Pocket-PC

by

Tony Scarpelli

ascarpe1@maine.rr.com

Don’t forget Random Access files!

You may think that ADOCE is the only way to create useful data tables, but not so. Before databases even existed for me, I was creating data tables in Basic using Random Access files. They are generally faster and fairly easy to create, but a little harder to work with because more work is needed to sort and search for things. Yet, I have found them a lot faster to work with on small Palm Size PC’s and Pocket-PC's than the ADOCE tables that can be created.

For instance, I have a medical equipment file that contains about 9817 records of 115 characters in a 1.2 meg file, yet I can find any device in this table in less than a second. The longest time taken is loading a grid with the information, and that takes less than a second. And with random access files, you can add records, sort in different ways, delete records, and do all the other database things you need to do. Not always easily, but working with ADOCE can also be a pain.

The other reason I'm using random access files is that the original data tables don't have to be converted by the device from Access to the type required on the device. A text file going to the device does not have to be converted so the transfer is fairly fast. Since I have to have updated files on the device on a weekly or sometimes daily basis, the conversion of a table the size I use can take a really long time. Look at these statistics:

FoxPro .dbf --> Access .mdb --> device .cdb, 75 min for a 1 meg file, et vice versa, definitely too long a process.

BioMstr.txt for device (~30 sec to create), 1,176,565 bytes.

WOFile.txt for Device (~3 min to create), 2273 Work Order records (just the latest work orders), 488,695 bytes with 100 character notes. For a complete history, this represents 20,972 records, 7,843,902 bytes on the device with 255 character notes. This easily fits on a 32 meg Pocket-PC. We use Compaq iPaqs.

If all files have been updated, AutoSync will finish loading these files in less than two minutes. If just Work Orders Due are updated, auto sync will finish in about 1/2 minute.

Need I say more?

So let’s start out with what a random access file is by looking at how one can be created from a FoxPro data file.

A delimited text file is usually a character file with each line in the file containing fields separated by some delimiter like a comma, or a tab character, with a CRLF (Carriage Return/Line Feed) at the end. A typical set of records might look like this:

“Tony Scarpelli”,”Anytown”,”ME”,”04123”,”207-123-4567”
“Jack Smith”,”Sometown”,”ME”,”04114”,”207-555-1234”

As you can see, each field can be a different length, or even empty as the case may be.

An SDF file is similar, but the fields are of fixed length. Each record, however, still consists of a single line with a CRLF at the end.

```
Tony Scarpelli  Anytown   ME04103207-123-4567
Jack Smith      Sometown  ME04103207-555-1234
```

I can create SDF files very easily using FoxPro. FoxPro data files are where all our equipment data are kept. I first create a temporary equipment table (BIOTEMP.DBF) that is filtered to remove non-active devices, is sorted on the equipment number, which is unique, and set up to only create fields that are going to be needed on the device. Then I create the SDF file from the temp table using this program line:

```
COPY TO BIOMSTR.TXT TYPE SDF FIELDS BIOTEMP.MFG, BIOTEMP.ID, BIOTEMP.MODEL,
BIOTEMP.SN, BIOTEMP.DESCR, BIOTEMP.DATEPM, BIOTEMP.DATESERV, BIOTEMP.PM_CODE
```

I imagine something similar can be done with Access or other database managers. The result is a file that contains all the records in a format similar to this:

```
Hewlett Packard    001647803A  Display   1008A04108 1985121219851212  12
Liebel Flarsheim   00191CSV    ESU       FM-11050   1993071619930716 102
```

I put the manufacturer first, then the five digit ID number, the model number, description, serial number, date tested, date serviced, and the PM test number. I put the manufacturer first because when the ID number is put first, the conversion has a tendency to remove the “0’s” from the beginning of the record. Putting the manufacturer first is normally not going to be a problem, however, since we are going to pick out the fields that we need from each record when it’s converted to a random access file anyway. We’ll show how later.

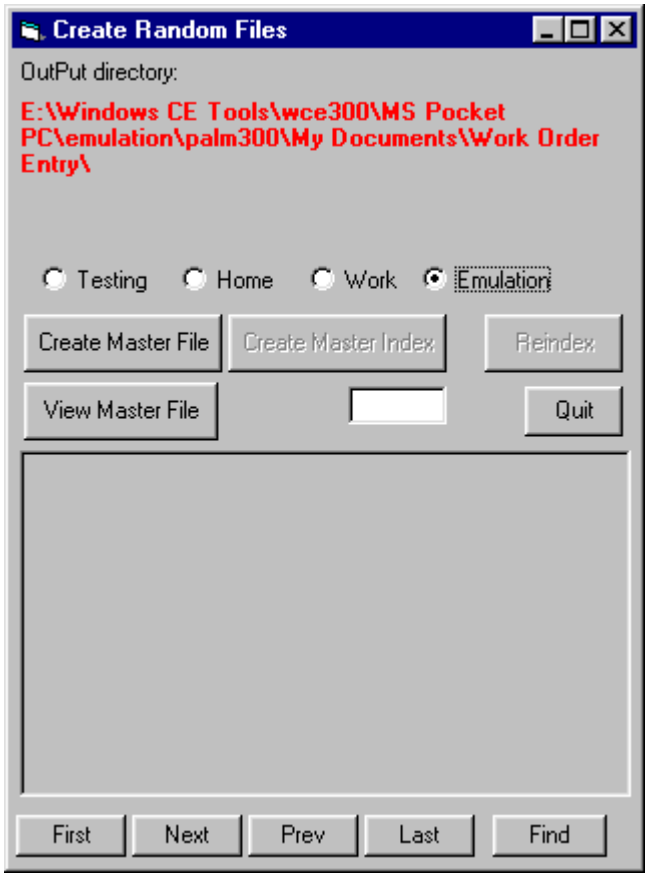
The major difference between an SDF file and a Random Access file is a four byte header at the beginning of each record, and there is no CRLF at the end of each line, it’s just one long string of characters:

```
****Hewlett Packard    001647803A  Display   1008A04108 1985121219851212
12****Liebel Flarsheim   00191CSV    ESU       FM-11050   1993071619930716 102
```

When we convert from an SDF file to a random access file all we have to know is the exact length of each record.

The program for the Work Station

This is my Visual Basic 6 form that creates the file:



Following is the Visual Basic 6 code to create the master text file from the SDF file, but before we get into the code, let me explain a couple things I do for my programs: all, or most of, my global variables begin with ‘g’, local variables begin with an ‘l’, character variables have a ‘c’, numbers have an ‘n’, arrays have an ‘a’, and I try to use upper and lower case for the names. I always make two constants in Basic, ‘t’ for ‘True’ and ‘f’ for ‘False’, this keeps the typing down and saves some space when programming on the device itself.

At the start of the VB code we explain what we are doing:

```
'This program creates the BIOMSTR.TXT random access
'file from the E:\BIOMSTRC.TXT SDF file.
'Make sure that you run the FoxPro program:
'BIOMSTRC.PRG file first to make
'sure that the SDF file is current.
```

Now we need a few variables set up:

```
Option Explicit

Private MstrCount, RecLen, glError, gnFileID
Private i, K As Integer, gaIDRec(9)
Private gcFileInPath, gcInFile, gcOutFile
Private Low, High, gcOutFilePath, gnFileNum
```

```
Private gnNumIDRecs, gnStartRec, gnEndRec
Private gnIDCnt, gcIDRec, gnNumRecs, gnTabNum
Private gcFoundRec, gnFoundRec, gnFileIndex
```

Then we start the Form up:

```
Private Sub Form_Load()
  'Close all files
  Close

  'Set paths for files
  SetOutPutPath
  gcFileInPath = "E:\"
  gcInFile = "BIOMSTRC.TXT"
  gcOutFile = "BIOMSTR.TXT"

  'Set length of master record
  RecLen = 140

  'Show form
  Show

End Sub
```

Because I work at home and at work, or if I just want to test the application, I have some option controls, **OptTest**, **OptHome**, **OptWork**, and **OptEmul** that I can set to have the output file put in a number of places.

```
Private Sub OptTest_Click()
  SetOutPutPath
End Sub

Private Sub OptHome_Click()
  SetOutPutPath
End Sub

Private Sub OptWork_Click()
  SetOutPutPath
End Sub

Private Sub OptEmul_Click()
  SetOutPutPath
End Sub
```

The default is to the root directory of one of my hard drives. But it can also go to the emulation directory which is available on my work NT workstation, or one of the sync directories. When placed in a sync directory, it automatically gets put onto the device when synced up.

This next bit of code sets the Path for our output files.

```
Private Sub SetOutPutPath()
  If OptTest.Value = True Then
    gcOutFilePath = "E:\"
  End If
  If OptHome.Value = True Then
    gcOutFilePath = "C:\My Documents\Pocket_PC_SCARP My Documents\Work Order Entry\"
  End If
```

```
  If OptWork.Value = True Then
    gcOutFilePath = "C:\WINNT\Personal\Pocket_PC_SCARP My Documents\Work Order Entry\"
  End If
  If OptEmul.Value = True Then
    'gcOutFilePath = "E:\Windows CE Tools\wce211\ms palm size pc\emulation\palm211\"
    gcOutFilePath = "E:\Windows CE Tools\wce300\MS Pocket PC\emulation\palm300\Program Files\Work
Order Entry\"
  End If
  lblMsg.Caption = gcOutFilePath
  frmCreateFile.Refresh
End Sub
```

This is the code behind the '**Create Master File**' button:

```
Private Sub cmdCreateFile_Click()
  CreateMasterFile
End Sub
```

Which calls this following routine.

```
Private Sub CreateMasterFile()
  lblMsg.Caption = "Creating Master file..."
  frmCreateFile.Refresh

  Dim lcFileName, lcFileOut
  Dim lnCnt, lnNumRecs
  lcFileName = gcInFile
  lcFileName = gcFileInPath & lcFileName

  Dim lcIDRec, lcThisLen, lcIDRec2, lcIDRec3
  Dim lcRec, lcRecLen, lnFileNum1, lnFileNum2
  lnCnt = 1
  lcIDRec2 = String(RecLen, " ")
  lcIDRec3 = String(RecLen, " ")
```

We open the input SDF file using the standard Basic OPEN statement:
Open *FileInName* For Input As #*1*, so we have to have an actual file name, and a file number that we can get from the **FreeFile** function which gets the next available file handle.

```
  On Error Resume Next
  'Open biomstr file
  lnFileNum1 = FreeFile

  Open lcFileName For Input As #lnFileNum1

  If Err <> 0 Then
    MsgBox "Error opening " & lcFileName & " data file." & vbCrLf & Err.Description, vbCritical, "Create
File"
    Err.Clear
    Exit Sub
  End If
```

Next we open the output file, which will become the random access file using the standard Basic random access syntax:
Open *FileOutName* For Random Access Write Lock Write As #2 Len = *RecordLength*

```
  'Open output file
  RecLen = 111 + 4
  lcFileOut = gcOutFilePath & gcOutFile
  lnFileNum2 = FreeFile
  Open lcFileOut For Random Access Write Lock Write As #lnFileNum2 Len = RecLen

  If Err <> 0 Then
    MsgBox "Error opening " & lcFileOut & " data file." & vbCrLf & Err.Description, vbCritical, "Create File"
    Err.Clear
    Exit Sub
  End If
```

If we didn't get any errors opening the files, we get to this point. Here we have a loop that pulls in each line of the input file using this statement:
Line Input #*filenumber*, *varname*
and displays the number on the form so we know that something is going on:

```
Do While Not EOF(lnFileNum1)
    'Get the biomstr record
    Line Input #lnFileNum1, lcIDRec
    lcIDRec2 = lcIDRec
    lblCnt.Caption = Str(lnCnt)
    frmCreateFile.Refresh
```

Then we save each random access record using this syntax:
Put [#]*filenumber*, [*recnumber*], *varname*

```
    'Save the record as random record
    Put #lnFileNum2, lnCnt, lcIDRec2

    lnCnt = lnCnt + 1
  Loop
```

We save the number of records we've created and close the files.

```
  MstrCount = lnCnt

  Close #lnFileNum1
  Close #lnFileNum2

  lblMsg.Caption = "Done!"
  frmCreateFile.Refresh
  Wait2Secs
  lblMsg.Caption = ""
  lblCnt.Caption = ""
  frmCreateFile.Refresh

End Sub
```

If you need to put in a little two second timeout, this is the little routine I use:

```
Private Sub Wait2Secs()
  Dim NextSec, NowTime
  NowTime = Time$
  NextSec = Val(Right$(Time$, 2)) + 2
  Do
```

```
  Loop While Val(Right$(Time$, 2)) < NextSec
End Sub
```

You can examine the output file using Notepad or Wordpad, but you should note that the header will display as some unknown ASCII characters, there won't be any CRLF at the end because each record is fixed in length, and the file will be just one long string.

My form also has a few extra controls on it so that I can create an index, reindex, and view the file, but we won't go into that in this article. Creating an index is handy, and adding a binary search makes finding random access records real fast. Perhaps we can go into that in another article.

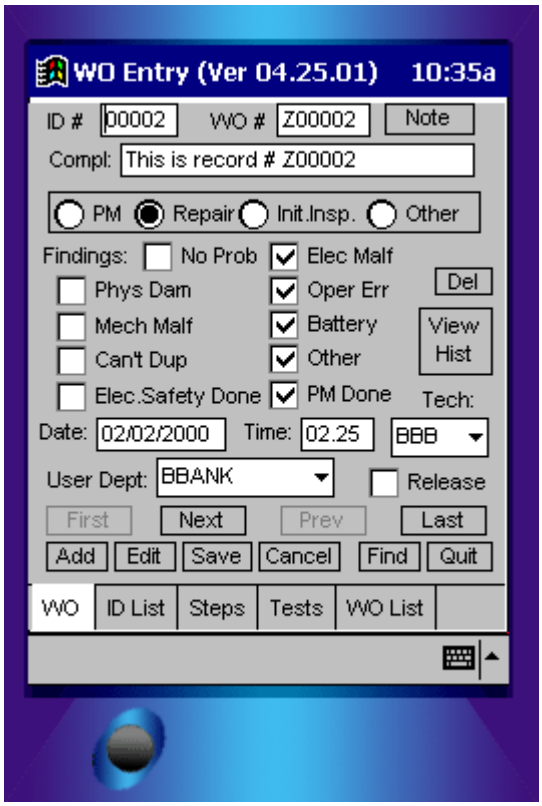
Now that we have created the file, and it resides on the device, what's next? We will now actually read in the records with a program that resides on the device. Next section.

The program for the Pocket-PC

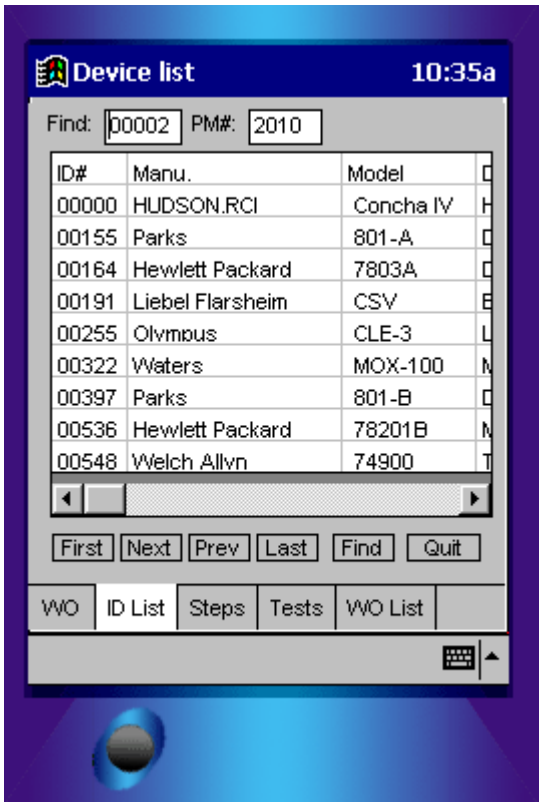
When the Random Access file was created above, it would have normally been put into the sync directory, and if so, will eventually end up on the device. If you have Windows NT or Windows 2000, you can also put it into the emulation directory for much faster testing. However, we really want the files on the Palm Size PC or Pocket-PC device when we are doing the real work of entering work orders.

When the Random Access file is on the device we have to open it, and then store the information from the random access records, into a grid.

The startup form for my application on the emulator looks like this:



The list of devices that comes from the Random Access file are placed in a grid and looks like this:



These are views using the emulator, the forms are very similar when actually on the device.

And the code for the device follows.

We start with some constants and global variables, then set the directory where the file ended up on the device.

```
Const t = True
Const f = False

Dim glSaved, glError, gcDirectory, gcFilePath
Dim gcFormCap, gcVer, gnNumIDRecs
Dim gnStartRec, gnEndRec, gcPM, gaIDRec(9)

'For device
gcDirectory = "My Documents\Work Order Entry\"
```

When the form loads we open the files, and if there was an error we exit, else we display the application title and version number, and put our cursor into the field we want it to be, in this case the equipment ID field.

```
Private Sub Form_Load()
  gcVer = "04.25.01"

  gcFormCap = "Opening files, standby..."
  frmBiomed.Caption = gcFormCap

  'Show form to see messages
  frmBiomed.Show

  'Open devices table
```

```
  OpenFiles

  If glError = t Then
    App.End
  End If

  gcFormCap = "WO Entry (Ver " & gcVer & ")
  frmBiomed.Caption = gcFormCap
  txtID.SetFocus

End Sub
```

This routine normally opens a lot of files, one after the other, but for this article we are only looking at the one we created above.

```
Private Sub OpenFiles()
  Dim lcFileName

  'Get path for file
  gcFilePath = gcDirectory

  gcFormCap = "Opening Master file, standby..."
  frmBiomed.Caption = gcFormCap

  OpenMaster

  If glError = t Then
    Exit Sub
  End If

End Sub
```

The above routine calls the following routine.

```
Private Sub OpenMaster()
  Dim lcAppPath, lcFilePath, lcFileName
  Dim RecLen, lnCnt, lnNumRecs, lnRecLen

  'Set path for files
  lcFilePath = gcDirectory
  lcFileName = "Biomstr.txt"
  lcFileName = lcFilePath & lcFileName
  RecLen = 111 + 4

  'Open the equipment table
  glError = False
```

The syntax to open a file on the device is as follows:
File.Open pathname, mode, [access], [lock], [reclength]
The **fsModeRandom, fsAccessRead, fsLockRead** variables are normally initialized at the beginning of the program, but we put in their values here for you to see. So we open the file and check for an error.

```
  On Error Resume Next
  'ceIDFile.Open lcFileName, fsModeRandom, fsAccessRead, fsLockRead, RecLen
  ceIDFile.Open lcFileName, 4, 1, 3, RecLen
```

```
  If Err.Number <> 0 Then
    MsgBox "Error opening " & lcFileName & " data file." & vbCrLf & Err.Description, vbCritical, "Work
Orders"
    Err.Clear
    glError = True
    Exit Sub
  End If
```

This next code determines how many records we have in the file with a little math, and then get the record length the same way. This I can use for checking the length of the record when testing.

```
  gnNumIDRecs = Round((celIDFile.LOF / RecLen))
  lnRecLen = (celIDFile.LOF / gnNumIDRecs)
```

Then we put the first few records into a grid on the form.

```
  'Load grid with data
  LoadIDGrid

End Sub
```

This following bit of code does two things: if the Find field has any data it, we are in a search mode where we look for the ID number. That code is not in this article. What we do set up, however, is what records we start and end at. Rather than load every single record into the grid, which slows the whole process down (we can have thousands of records), we only load enough to show on the screen in the grid. Then, if you notice on the form, we can use navigate buttons to move through these records simply by incrementing and decrementing **gnStartRec** and **gnEndRec**.

```
Private Sub LoadIDGrid()
  'Load grid with ID data

  If Trim(txtFind.Text) = "" Then
    gnStartRec = 1
    gnEndRec = 9
  Else
    'FindID
  End If

  GetIDs

End Sub
```

This following code first sets the grid up. I won't cover every line since they are all in the help files on grids. Then we set a For/Next loop that gets each record (**GetIDRec**), and puts each field into an array, and then into the type of line the grid wants.

```
Private Sub GetIDs()
  'Fill the grid with the contents of the Table
  Dim strRow, lnColumns, i

  'Reset grid first
  grdData.Clear
  grdData.Rows = 0
  grdData.Cols = 0
  grdData.ScrollBars = flexScrollBarHorizontal
```

```
'Set # of Columns
 lnColumns = 9
 grdData.Cols = lnColumns
 'grdData.CellAlignment = 9

 'Set Column width
 grdData.ColWidth(0) = 570
 grdData.ColWidth(1) = 1600
 grdData.ColWidth(2) = 1000
 grdData.ColWidth(3) = 2000
 grdData.ColWidth(4) = 1500
 grdData.ColWidth(5) = 1000
 grdData.ColWidth(6) = 1000
 grdData.ColWidth(7) = 570
 grdData.ColWidth(8) = 1400

 grdData.ColAlignment(0) = flexAlignLeftTop
 grdData.ColAlignment(1) = flexAlignLeftTop
 grdData.ColAlignment(2) = flexAlignLeftTop
 grdData.ColAlignment(3) = flexAlignLeftTop
 grdData.ColAlignment(4) = flexAlignLeftTop
 grdData.ColAlignment(5) = flexAlignLeftTop
 grdData.ColAlignment(6) = flexAlignLeftTop
 grdData.ColAlignment(7) = flexAlignLeftTop
 grdData.ColAlignment(8) = flexAlignLeftTop

 'Add a row for column headers
 strRow = "ID#" & Chr(9) & "Manu." & Chr(9) & "Model" & Chr(9) & "Description" & Chr(9) & "Serial #" &
Chr(9) & "Date PM" & Chr(9) & "Date Serv" & Chr(9) & "PM #" & Chr(9) & "JFMAMJJASOND"
 grdData.AddItem strRow

 'File is OK, get all the records
 For i = gnStartRec To gnEndRec

   'Read the record
   gnIDCnt = i
   GetIDRec
```

Notice below that the ID number, **galDRec(1)**, which is in the second field position, is now displayed in the first position. And also note that arrays will start with 0.

```
   'Create the row
   strRow = ""
   strRow = strRow & galDRec(1) & Chr(9)
   strRow = strRow & galDRec(0) & Chr(9)
   strRow = strRow & galDRec(2) & Chr(9)
   strRow = strRow & galDRec(3) & Chr(9)
   strRow = strRow & galDRec(4) & Chr(9)
   strRow = strRow & galDRec(5) & Chr(9)
   strRow = strRow & galDRec(6) & Chr(9)
   strRow = strRow & galDRec(7) & Chr(9)
   strRow = strRow & galDRec(8)

   grdData.AddItem strRow

 Next
```

End Sub

The following code is really where we reverse the process: retrieving the data from the random access record. We get one record from the random access file, separate each field out, and put them into an array. You may ask why not put the fields directly into the grid. Well, you could, but arrays are good to work with, they are fast, and they're even better when debugging.

```
Private Sub GetIDRec()
  Dim lcIDRec
```

These next few lines basically show how my device record is set up. If you remember from above, what we have is the manufacture's name, the device's ID number, the model, description, serial number, and some dates and a bit of other information we need. If you display the record this way, you can easily tell how large each field is and where it starts; it's a big help during development to look at the record this way. By the way, you want to use a fixed length font such as **Courier New** when doing this. Don't even bother trying it with something else. Because the data is longer than this page, some information at the end is truncated off.

```
'                                                                                                                  1
'         1         2          3         4         5         6         7         8          9          0
'12345678901234567890 12345 6789012345 67890123456789012345 678901234567890 12345678 90123456 789 0
'12345678901234567890 12345 1234567890 12345678901234567890 123456789012345 12345678 12345678 123 4
'Hewlett Packard      00164 7803A      Display                       1008A04108      19851212 19851212   1 2
```

This next bit of code gets the individual record from the file. It will take the single line, like the last line above, and puts the record into the variable **lcIDRec**.

```
  'Read the record
  ceIDFile.Get lcIDRec, gnIDCnt

  If Err.Number <> 0 Then
    MsgBox "Error Getting record" & vbCrLf & Err.Description, vbCritical, "Error Encountered"
    Err.Clear
    glError = True
    Exit Sub
  End If
```

We now put that line into a global variable so the next routine can see it. This code also does some converting of the data. And finally puts the field data into a global array with the **LoadIDArray** routine.

```
  'Save to global
  gcIDRec = lcIDRec
  gcPM = "2" & PadL(Mid(lcIDRec, 97, 3), 3, "0")
  txtPM.Text = gcPM

  'Load array with data from string
  LoadIDArray

End Sub
```

This code will pull each field into the array by using the Mid() function. As you can see, this is why the formatted lines above help with this bit of code; we know where the field starts and how long it is.

```
Private Sub LoadIDArray()
  'Load array with data from string
  gaIDRec(0) = Mid(gcIDRec, 1, 20)   ' Manufacturer
  gaIDRec(1) = Mid(gcIDRec, 21, 5)   ' ID #
  gaIDRec(2) = " " & Mid(gcIDRec, 26, 10)  ' Model
```

```
  galDRec(3) = Mid(gclDRec, 36, 30)  ' Description
  galDRec(4) = " " & Mid(gclDRec, 66, 15)  ' Serial #
  galDRec(5) = Mid(gclDRec, 85, 2) & "/" & Mid(gclDRec, 87, 2) & "/" & Mid(gclDRec, 81, 4)  'Date PM
  galDRec(6) = Mid(gclDRec, 93, 2) & "/" & Mid(gclDRec, 95, 2) & "/" & Mid(gclDRec, 89, 4)  'Date Serv
  galDRec(7) = Mid(gclDRec, 97, 3)  'PM #
  galDRec(8) = Replace(Mid(gclDRec, 100, 12), " ", ".")  'PM Code

End Sub
```

That's it.

Since we are always working with text files and character strings, the code is very fast. This is very important in a hand held device with a slow or slower processor. So if you need this kind of speed, forget database files in Access and go with Random Access.