FileIO Presentation

During the first part of this presentation, we're going to cover some of the FileIO basics for anyone who may be new to FileIO, while taking you through an example of beginning to implement and use fileio with existing data files.

As long as you don't actually change the file layout without continuing to update your older non-fileio programs, you can use FileIO to save time and increase future maintainability as you continue to develop new programs.

For this example, we're going to use a sample Customer file, which I have supplied in the presentation materials. Imagine that this data file is part of an existing software application. In the next few minutes, I'm going to take you through the process of creating a file layout for this file, and using the FileIO library to write a simple report program that loops through the file, printing out each customer.

You can download the sample environment for this tutorial here:

Sample Environment

The sample customer data we're using for this example is taken from our database of BR vendors. The addresses have been removed and all the phone numbers have been changed.

Part 1: Create File Layout

The first thing to do is create a file layout. This process is very simple. As simple, in fact, as creating a text file and typing your file layout into it.

To make it even easier, I often simply copy an existing file layout, rename it, and modify it as necessary. But, for now, lets take a look at the process of creating one from scratch, and we'll take a closer look at each section of a file layout and what it does.

Because many of you already have your file layouts in text files, you may be able to just copy and paste them into FileIO layouts, making modifications where necessary. But if not, its still a fairly simple matter to type them in from whatever format you keep them in.

Begin by gathering all the information you have about the structure of the file layout. In this example, we're using the customer data file, and I'm going to tell you what to put in the file layout, so simply browse to or create a folder inside your application called "fileio" and create a new text file. Open that file in your favorite text editor and you're ready to begin.

A FileIO file layout consists of a header and a detail section. The header describes the filename and path, the version number, any key files, and the record length. The detail section lists all the fields in the data file, and their field types.

Lets take a look at the header section first.

```
   customer.dat, CU_, 8    customer.key, CODE    recl=127
================================================
```

A file layout is a CSV file. The first line has three elements on it. The first is the file name and path. The second field you see is a unique prefix associated with the file. We will use this prefix to identify fields from the file in our programs. The third field indicates the current version of the file.

FileIO will automatically update your data file if this version number is higher then the current version of the file. If you don't want fileio to update your data file, then set the version in the file layout to the correct version matching the one on the disk. Its easy to tell what version your data file is. Simply run one of your programs that opens the file, and pause the program with the file open. Then type Status Files at the prompt and it will tell you most of the information you need to make your file layout header.

The File Layout for the customer file has a filename of customer.dat. Its in the current directory. It has a unique prefix of CU_, and the file is at version 8.

The next several lines of the file layout describe the keys that the file uses, if any. Specify each key on its own line, followed by the field names tha the keys are made out of. We'll take a closer look at field names in a minute.

The customer file in this example has one key, called customer.key, which is build off the Customer Code field, which we're going to name "CODE".

After all your keys are specified, you need a line telling the record length of your file. It looks like the one in our example. This sample file has a record length of 127.

After that we have one more row with no commas, which is ignored by FileIO. Use this row to make the file layout more readable by placing a horizontal divider between the header and the detail sections.

Now that we've gone over all the parts of the header, lets take a look at the detail section of the file layout.

```
   CODE$,        Customer Code,             C   4   NAME$,        Customer Name,             V
30   CITY$,        Bill To City,              V  30   ZIP$,         Bill To Zipcode,           C   5
PHONE$,       Phone Number,              C  10
```

The sample file has five fields in it. Each row in the layout describes another field, and there are three elements on each line. The first one is the name of the field. This is the same name that's used to describe the keys above. This name is also used in code whenever you want to reference this field in the data file. The second element is a short description of the field. This could be a description you would show to the end user on a screen for maintaining the field. The third element is the form spec of the field on the disk. This should be familiar to anyone who is programming in BR.

The customer file has a uniquely generated customer key, a Customer Name, City, Zipcode, and Phone Number. Add these fields to your layout now, or copy and paste them from this document.

Save the file, and you're ready to test it.

Using the Data Crawler to Test your new File

Now that you have your file layout created, you immediately gain access to a whole host of tools, many of which are free, that you can now use to help manage your software suite more effectively and productively. The easiest to use of them is called the Data Crawler. Its a simple BR program that will view the contents of any of your BR data files. It is very useful as a debugging and development tool.

We're going to use the data crawler now to test the new file layout we just created. Launch a copy of BR, load FileIO, and run it. The Data Crawler opens, presenting you with a listview showing all the file layouts defined in your system. This list shows all the layout files found in the filelay folder. Select the customer file now.

If you get an error, or if you only see partial data, there's probably something wrong with the file layout. Review the file layout looking for a missing comma or missing information. The finished file layout should look like this:

```
   customer.dat, CU_, 8    customer.key, CODE    recl=127
================================================    CODE$,         Customer
Code,             C    4    NAME$,         Customer Name,             V   30    CITY$,         Bill To
City,             V   30    ZIP$,          Bill To Zipcode,           C    5    PHONE$,        Phone
Number,           C   10
```

Once you've fixed any errors that may be in the file layout, you should be able to run the data crawler again, and view the contents of the customer.dat sample data file.

Once you get your file layouts created, even if you don't use FileIO for anything else, you'll find the Data Crawler is an indispensable tool that will come in handy time and time again.

Using the FileIO Library

Its probably worth it to make fileio file layouts for your data files just so that you can use the DataCrawler. However, the FileIO library can do so much more for you then just that.

Lets take a look at a simple report program that uses fileio to print a customer list report.

```
01000 ! CustRept.br   #Autonumber# 1000,10 01010 ! 01020 ! Created 3/30/2010 01030 ! by Gabriel Bakker 01040 ! 01050 ! Sample Library using FileIO to build a simple Customer Report. 01060 !
```

Every program should start with a comments section describing what it is and when it was made.

The #Autonumber# Comments you see everywhere are part of another free tool, Lexi. You can ignore them for now, but take a look at Lexi if you haven't already. Its another very powerful programming tool offered for free by Sage AX in order to help keep the BR market alive.

```
10000 ! .! #Autonumber# 10000,10 10020 ! 10030 ! . ! Step 1: Establish Library Linkage to FileIO 10040 library "fileio" : FnOpenFile 10050 !
```

The first step is pretty straightforward. Simply declare the library linkage to the fileio library. Remember to include any functions you will be using from the fileio library. In this program, the only function we're using is fnOpenFile.

```
10060 ! . ! Step 2: Declare File Object Variables, and Forms Array. 10070 dim Customer$(1)*255,Customer(1) 10080 dim Form$(1)*255
```

For every data file you intend to read with the fileio library, you will need to declare two arrays to store the information, one string array and one numeric array. Anyone familiar with LORICA or any of several other programming toolsets may be familiar with this concept.

If you need to keep track of multiple records of information simultaneously, you can declare additional sets of arrays to store them in.

```
10090 ! 10100 ! . ! Step 3: Open Data File 10110 let CustomerFile=fnOpen("customer",mat Customer$,mat Customer,mat Form$,1) 10120 ! 10130 ! . ! Also open the printer for printing a report, and print a heading 10140 open #255: "name=preview:/, recl=500",display,output 10150 print #255, using ReportFormSkip0 : "Customer Name","City","Zip","Phone" 10160 print #255, using ReportForm :
```

"_____",
"_____","_____","_____" 10170 !

Here we come to the meat of the fileio library: The fnOpen function. This function will read your file layout, open the data file, redimension your record pointer arrays to the correct size, and define subscript variables to help you reference the data from the file in your code.

This is a fairly standard call to fnOpen. The 1 at the end tells FileIO that we're opening the file for Input. This accomplishes several things, all of which you can read about in more detail the fileio documentation.

The FileIO Open function finds the first available file number and uses it to open the file, returning the file handle so you can use it to access the file.

The FileIO library is designed to be as minimally invasive as possible. All the magic happens in the open statement. From then on you read the data file the same way you always have, with the BR read statement.

Since we're writing a report program, we also need to open the printer and print the report headings.

10180 ! . ! Step 4: Read the Data File in a loop 10190 do while file(CustomerFile)=0 10200 read #CustomerFile, using form$(CustomerFile) : mat Customer$, mat Customer eof Ignore 10210 ! . ! Print the record 10220 let Customer$(cu_phone)="("&Customer$(cu_phone)(1:3)&")"&Customer$(cu_phone)(4:6)&"-"&Customer$(cu_phone)(7:10) 10230 print #255, using ReportForm : Customer$(cu_name), Customer$(cu_City), Customer$(cu_zip),Customer$(cu_phone) 10240 loop 10250 ! 10260 ReportFormSkip0: form C 30, C 30, C 7, C 13, Skip 0 10270 ReportForm: form C 30, C 30, C 7, C 13 10280 !

When you read a data file in fileio, you read the file using the File Access Arrays we created earlier. FileIO calculates the form statement for you and places it in the FORMS$ array at the same position as the file number for the data file.

The read loop above will read every record in the data file looping until we hit the end of the file. For each record read, we print it out to the printer.

Notice how we read the data a record at a time using the arrays, and we never specify individual variables in the read statement. Instead, we access the fields by name in the arrays using our named subscripts.

Using fileio, you are free to change the file layouts for any of your files at any time. Because your programs always use the subscripts and the arrays to access the fields by name instead of by position, you can change your file layouts, adding new fields, inserting old fields, moving stuff around and rearranging it to your hearts content. All your programs that use FileIO will continue running like they always have.

```
 10290  ! .   ! Step 5: Close the Datafile, end the program  10300        close #CustomerFile:
10310     stop  10320  !
```

Since we opened the file Input Only, we can close it the same way as we normally would. However, if you open a fileio data file for Input/Output access, it automatically opens all your key files for you. Therefore, it is necessary to close the file using the fileio fnCloseFile routine. More information on this is available in the fileio documentation.

```
 10330  ! Step 6: Copy and paste standard FileIO Open Function.  99000  ! #Autonumber#
99000,10  99010  OPEN: ! ***** Function To Call Library Openfile And Proc Subs  99020     def
Fnopen(Filename$*255, Mat F$, Mat F, Mat Form$; Inputonly, Keynum, Dont_Sort_Subs,
Path$*255, Mat Descr$, Mat Field_Widths,___,Index)  99030        dim _FileIOSubs$(1)*800
99040        let Fnopen=Fnopenfile(Filename$, Mat F$, Mat F, Mat Form$, Inputonly, Keynum,
Dont_Sort_Subs, Path$, Mat Descr$, Mat Field_Widths, Mat _FileIOSubs$)  99050        for
Index=1 to udim(mat _FileIOSubs$) : execute (_FileIOSubs$(Index)) : next Index  99060
fnend  99070  !  99900  ! #Autonumber# 99900,10  99910  IGNORE: continue
```

The last step is to copy the standard FileIO fnOpen Wrapper Library into all of your programs that will be using fileio. This wrapper function is necessary in order for fileio to create the subscript variables that you use to access the data in the fields in your file. Simply copy and paste the function from the bottom of the fileio library itself.

That's all there is to it. Using FileIO saves future maintenance and development time, while helping keep your code structured and readable. FileIO also gives us a standard language we can use to create powerful tools that build off of FileIO. We're going to take a look at several of these tools next, including The ScreenIO Library.