

Launching into BASIC

Joe Abernathy

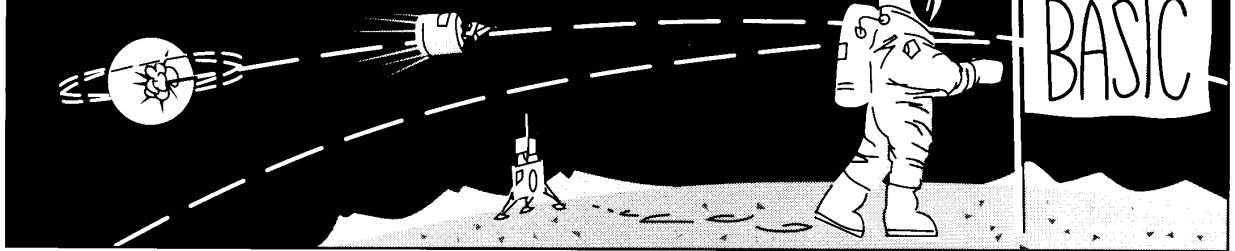


Illustration by Robert Williamson

Top programmers rely upon a firm command of data structures in order to turn out successful programs. We will show how you can add this polish to your applications, from simple variables to Toolbox pointers and handles.

This month and next, Launching Into BASIC will discuss the full range of Apple IIGS data structures in the context of real world examples. In this installment, we examine the data structures common to AC/BASIC and TML BASIC and show how to build a library of routines with which you can manage the environment for a customer database. Next month, we implement this database library in TML BASIC and examine the advanced data structures offered by TML.

The compilers of today stretch the old idea of "data structure" quite a bit. You now can assign local variables in order to reduce program clutter; extract information by bytes from a special array type; and use memory pointers to access the internal Apple IIGS tools.

Despite the edge it gives, there is little to learn to master professional data manipulation. Particularly where pointers and handles are concerned, there is an unjustified perception of difficulty.

Built-in Data Facilities

Any logical representation of information is a data structure, including our database files. Built-in facilities include such things as variables and arrays, which are a part of the language. There are few differences between how Absoft and TML implement data handling. These will be noted when appropriate.

Variables — Named variables can be assigned a value either directly; through the result of a mathematical operation; or through the returned value of a function.

Scratch

Does anyone know the origin of the word "scratch" as applied to computers? In Joe's use here, it means a variable that is available for use on a temporary basis. It has no permanent role, it is used like a piece of scratch paper. It can be used for different things at different times.

I first ran across the term when using tapes back in the old days when they were a foot in diameter. If we needed a tape to temporarily store data, e.g., a sort usually needed a couple of scratch tapes for temporary files, there was a shelf of scratch tapes. Once we were done with a tape, we put it back on the shelf of scratch tapes for anyone to use. Of course, if the tape was used for something like a backup, it went to the permanent shelf.

Cecil

Global variables are available throughout a program. They can be assigned at startup, through calling a file of standard assignments, and by subroutines. You can declare a variable as local in order to hide it within a subroutine. This is especially valuable in large programs where variable overwrites can be a problem.

Numeric variables allow four types. They are integer, single precision, double precision, and extended precision. For most math, the integer variable type will provide sufficient precision and the greatest speed. The more sophisticated types allow you to do scientific math and access SANE, the Standard Apple Numerics Environment.

You will notice that I sometimes use scratch variables such as x\$ when a locally declared variable would be the right way to do it. My habit is a holdover from languages in which the lack of local variables forced you to use a pool of scratch variables in order to maintain structure. You will likely want to adopt a new habit of declaring scratch variables locally. (See Scratch sidebar.)

Variable examples:

```
LET x = 10

LOCAL x$
x$ = "Sid"

x% = FN_GetAge (birthyear%, year%)
```

Arrays — Arrays allow you to group related information. You can use any legal variable type for an array declaration. The only restriction is that all of its elements must be of that type. An array can have more than one dimension (up to 8 dimensions with TML, limited by memory in AC/BASIC).

Elements of arrays can receive and pass values just as with simple variables. A special array type in TML BASIC lets you manipulate information on the byte level. This "struct" array will be discussed next month when we develop records.

Arrays are often used for tasks such as loading lists of information from disk. The second example below, which is an extract from the datafile listing, shows how this can be done. Note that in TML BASIC, 0 is the first element of the arrays, so we use the loop counter minus one as our index.

Examples:

```
DIM DYNAMIC Scores$(NumTests%, NumPupils%)

DIM fwCustList$(NumCustomers%)
REM load list of customer files
DIM fwName$(NumCustomers%)
DIM fwPhone$(NumCustomers%)
```

```

FOR i = 1 to NumCustomers%
  x$ = fwCustList$(i-1)
  OPEN x$, FOR INPUT AS #FileIndex%
  INPUT #FileIndex%; y$
  fwName$(i-1) = y$
  INPUT #FileIndex%; y$
  fwPhone$(i-1) = y$
  CLOSE #FileIndex%
NEXT i

```

Records — There is no AC/BASIC data structure that correlates to the Pascal record. Next month, however, we will show how Pascal programmers can utilize their knowledge of records with the TML BASIC struct array type.

Pointers — AC/BASIC insulates the programmer from the manipulation of memory pointers. This data structure type, most often used when issuing Toolbox calls, will be addressed next month under the auspices of TML BASIC.

Handles — Memory handles, too, will not confront you under AC/BASIC. Handles allow the Memory Manager to do its job, dynamically reassigning memory locations to blocks of code at run time. We will discuss pointers next month.

Data typing — Another data facility that you should be aware of although it is not yet available in any BASIC for the IIGS. With data typing, you can create data structures dedicated to a particular type of information such as phone numbers or credit card accounts. I suspect that some form of data typing will make its way to us in the summer of 1989.

Custom Data Facilities

Since data files are just another way to arrange information, they are considered to be data structures. You can use data files for a variety of housekeeping jobs to add elegance to your work. We will demonstrate them by implementing a library used to manage a customer database.

Data files, or at least the internal structure of data files, have been the source of occasional controversy. The rights to a useful data file are rightly regarded by vendors as valuable property that should be protected. So how then does an independent programmer go about reading a BusinessWorks company file, for instance, when the file structure isn't published?

Apple Developer Services has attempted to set file format standards with its most notable success coming in the area of picture files. This success is a reflection of the graphics tools built into the IIGS. Since these tools have no text file counterparts, you have the opportunity to create a data file template conforming to your own specifications.

To obtain the most from your data files, you should create a format general enough to be used by more than one program. By studying Listing 2 as an example, you can see how to create both a master data file and a set of customer information files which can be used by any program written in any language. Indeed, these DataFileLib structures are a scaled down implementation of a file format I have used in three commercial programs, each written in a different language.

This demonstrates a sequential access text file in which lines of information are read in sequential order. Another type of file access, random access, is much more efficient, but for the application being developed here, it would be difficult to implement and more limited in its utility. If you decide to expand your data file library, random access routines would be a good place to start.

When you want to load the information stored in a data file, an array often is the best choice. However, for such things as

values read from the master data file, simple global variables might be more practical. We will demonstrate both methods.

The AC/BASIC Data Library

The data file library implements a set of procedures that collectively manage the disk and data structures of a customer database. It comprises five procedures: FWDDataSetUp, FWWriteCust, FWDeleteCust, FWShutDown and UpdateFWDData.

Listing 1 is an AC/BASIC program used to demonstrate the functionality of the library. Listing 2 is the actual implementation of the customer database.

Missing are the calls to implement a user interface to the library. This communication with the user, even under the desktop metaphor, tends to be different in every application. (We will, however, discuss data input dialogs in the future.)

It is quite easy to implement the database library from your program shell. Under AC/BASIC, just make the additions shown in Listing 1. Remember to add the new values under SetUp to your shell's SetUp label.

The procedures:

FWDDataSetUp — This procedure prepares the customer database for use. It reads two internal data files to determine the number of customers online and their names. It then correlates each name into a file name, dimensions the customer data arrays, and loads the customer database information into these arrays.

The first time FWDDataSetUp is called, it will create its master data files on the drive from which its calling program was booted. These include “*/SYSTEM/FW.DATA”, “*/SYSTEM/CUSTOMERS/”, “*/SYSTEM/CUSTOMERS/CUST.LIST”, and the file called “*/SYSTEM/CUSTOMERS/ABERNATHY.JOE” for the default first customer.

Advanced Study: Implement a “Please wait ...” dialog to be displayed if more than 30 customer files must be loaded at startup. You will need a separate “end wait” call to remove the dialog from the screen. (You can modify the “MsgDialog” tool.)

FWWriteCust(whom%) — This procedure either adds a new customer to the database or updates information concerning an existing customer. If you pass a zero value in whom%, the information at the upper bound of the customer data arrays will be used to add a customer. If you pass a non-zero value, the information for that customer number will be updated with the information at the matching offset in the customer arrays. No error checking will be performed to ensure that this is what you really want to do.

When you put this routine to use, it will usually be called from a procedure that obtains customer information either from the keyboard or simple variables. In order for FWWriteCust to use your new customer information correctly, the calling routine must always load the customer arrays at their upper bound.

FWDeleteCust(which%) — This procedure removes a customer from the database. Which% should be a valid customer number. No error checking will be performed to ensure that this is so.

If you pass a zero value for which%, FWDeleteCust will return with all data unchanged.

FWShutDown — Normally, this call would release array and other memory used by the data file library. In AC/BASIC, you must ERASE arrays in the main program (as discussed in the source code comments) so it amounts to a marginal call

whether you should bother with FWShutDown. This call will be more useful in its TML BASIC incarnation.


UpdateFWData — This procedure might eventually wind up in several different programs you write. When you share data files, remember that changes and additions to the fields made by one program will affect all programs.

Advanced Study: Implement a NumFields% value in the FW.DATA master file to allow dynamic changes to the number of subsequent fields read. Hard wire a hierarchy governing the order in which variables are read. Any variables read past the end of the data meaningful to a particular application should be discarded by that application.

Resources

We welcome your questions, complaints and kudos. Write to Launching Into BASIC in care of *Call -A.P.P.L.E.* On AppleLink — Personal Edition, send E-mail to JOEA17.

Questions for the column and for the *Call -A.P.P.L.E.* Consultants' Corner also are accepted in The Cortland Project SIG on Europa bulletin board system. Launching Into BASIC listings are available for download: (713) 526-0714, 1200/2400 bps.

Call -A.P.P.L.E. listings also are available on the TechAlliance Bulletin Board System (TBBS, formerly Apple Crate) approximately one month after publication: (206) 251-6775 or 251-6784. 

Listing 1

```

-----
' This is a short demonstration program to
' show how the datafile library might be
' called from AC/BASIC. Ellipses ... indicate
' a broken line.

' '$Include "DataFileLib"

GOSUB Setup
ON ERROR GOTO DataErr
OPEN "*" /SYSTEM /FW.DATA" FOR INPUT...
... AS #FileIndex
CLOSE #FileIndex
FWDataSetUp

CallDemo:
ON ERROR GOTO 0 ' Error handler off
GOSUB Demo
END

Demo:
PRINT
PRINT "There are "; NumCustomers; "...
... customers now on-line."
IF NumCustomers <> Nil THEN
PRINT
PRINT "Here is a screen to show...
... that the data library works:"
FOR i = 1 TO NumCustomers
x% = i - 1
PRINT
PRINT "Customer # "; fwCustNum$(x%)
PRINT "Name : "; fwFirstName$...
... (x%) + " " + fwLastName$(x%)
PRINT "Birthday : "; fwMbirth$...
... (x%); "-" ; fwDbirth$(x%); ...
... "-" ; fwYbirth$(x%)
PRINT "Social Security # "; ...

```

50 Call -A.P.P.L.E.

```

... fwSSN$(x%)
PRINT "Company : "; fwCompany$(x%)
PRINT "Address1 : "; fwAddr1$(x%)
PRINT "Address2 : "; fwAddr2$(x%)
PRINT "City : "; fwCity$(x%); ...
... " "; fwState$(x%); ...
... " "; fwZip$(x%)
PRINT "Phone : (" ; fwAreaCode$...
... (x%); " ) "; fwPhone$(x%)
PRINT "Last Contacted On : "; ...
... fwLastContacted$(x%)
PRINT "Comments : "; fwNotes$(x%)
NEXT i

ELSE
PRINT
PRINT "No customer files to read."
END IF
PRINT
INPUT "Press RETURN to exit demo: "; i$
FWShutDown ' Shut down this lib
RETURN

DataErr:
CLOSE #FileIndex ' Close test file
DoDataInstall
RESUME CallDemo

Setup:
DataReady = 0
Nil = 0
FileIndex = 1
NumCustomers = 0
RETURN

SUB DoDataInstall SHARED
NumCustomers = 1
DoDataDim ' Make arrays
MakeDataFiles ' Internal files
GetDefaultData ' First customer
FWWriteCust(1) ' Save customer
DataReady = 1
END SUB

```

Listing 2

```

-----
' File: DataFile.BAS
' Version 00000001 - 12/12/88
' Customer data file management library
' (C) Joe Abernathy. All Rights Reserved.
' Data files (C) 1986-89, First Word.
' All Rights Reserved. Used with permission.
' Compiler: AC/BASIC and compatibles
' Compile with N option set
' -----
' Contains: FWDataSetUp, FWWriteCust,
' FWDeleteCust, FWShutDown, UpdateFWData
' -----
' SUB FWDataSetUp
' Start up the customer data library:

SUB FWDataSetUp SHARED
OPTION BASE 0 ' For arrays
x1$ = "*" /SYSTEM /FW.DATA"
x2$ = "*" /SYSTEM /CUSTOMERS"
x3$ = "*" /SYSTEM /CUSTOMERS /CUST.NAMES"

' Has FWDataSetUp been called?
' If not, begin:
IF DataReady = Nil THEN
FileIndex = FileIndex + 1
' Read main data file:
OPEN x1$ FOR INPUT AS FileIndex

```

```

LINE INPUT #FileIndex, x$
NumCustomers = VAL(x$)
CLOSE #FileIndex
' Max # of new entries, 50:
x% = NumCustomers + 50
DIM fwCustNum$(x%)
DIM fwCustList$(x%)
DIM fwFirstName$(x%)
DIM fwLastName$(x%)
DIM fwInitial$(x%)
DIM fwYBirth$(x%)
DIM fwMBirth$(x%)
DIM fwDBirth$(x%)
DIM fwSSN$(x%)
DIM fwCompany$(x%)
DIM fwAddr1$(x%)
DIM fwAddr2$(x%)
DIM fwCity$(x%)
DIM fwState$(x%)
DIM fwZip$(x%)
DIM fwAreaCode$(x%)
DIM fwPhone$(x%)
DIM fwLastContacted$(x%)
DIM fwNotel$(x%)
ReadCustList      ' Read Customers
ReadCustData      ' Fill arrays
DataReady = 1
FileIndex = FileIndex - 1
END IF
END SUB

' Private procedure - Create internal
' customer data files:
SUB MakeDataFiles SHARED
OPEN "*/SYSTEM/FW.DATA" FOR OUTPUT...
... AS FileIndex
PRINT #FileIndex, NumCustomers
CLOSE #FileIndex
OPEN "*/SYSTEM/CUSTOMERS/CUST.LIST"...
... FOR OUTPUT AS FileIndex
Whom$ = "*/SYSTEM/CUSTOMERS/...
...ABERNATHY.JOE"

' Your first customer:
PRINT #FileIndex, Whom$
CLOSE #FileIndex
END SUB

' Private procedure - Read list of customer
' data file names
SUB ReadCustList SHARED
OPEN x3$ FOR INPUT AS FileIndex
FOR i = 1 to NumCustomers
LINE INPUT #FileIndex, x$
fwCustList$(i-1) = x$
NEXT i
CLOSE #FileIndex
END SUB

' Private procedure - Load customer arrays
' with proper info
SUB ReadCustData SHARED
FOR i = 1 to NumCustomers
x$ = fwCustList$(i-1)
OPEN x$ FOR INPUT AS FileIndex
LINE INPUT #FileIndex, y$
fwCustNum$(i-1) = VAL(y$)
LINE INPUT #FileIndex, y$
fwFirstName$(i-1) = y$
LINE INPUT #FileIndex, y$
fwLastName$(i-1) = y$
LINE INPUT #FileIndex, y$
fwInitial$(i-1) = y$
LINE INPUT #FileIndex, y$
fwYBirth$(i-1) = VAL(y$)
LINE INPUT #FileIndex, y$
fwMBirth$(i-1) = VAL(y$)
LINE INPUT #FileIndex, y$
fwDBirth$(i-1) = VAL(y$)
LINE INPUT #FileIndex, y$
fwSSN$(i-1) = y$
LINE INPUT #FileIndex, y$
fwCompany$(i-1) = y$
LINE INPUT #FileIndex, y$
fwAddr1$(i-1) = y$
LINE INPUT #FileIndex, y$
fwAddr2$(i-1) = y$
LINE INPUT #FileIndex, y$
fwCity$(i-1) = y$
LINE INPUT #FileIndex, y$
fwState$(i-1) = y$
LINE INPUT #FileIndex, y$
fwZip$(i-1) = y$
LINE INPUT #FileIndex, y$
fwAreaCode$(i-1) = VAL(y$)
LINE INPUT #FileIndex, y$
fwPhone$(i-1) = y$
LINE INPUT #FileIndex, y$
fwLastContacted$(i-1) = y$
LINE INPUT #FileIndex, y$
fwNotel$(i-1) = y$
CLOSE #FileIndex
NEXT i
END SUB

' Private procedure - Generate the first
' customer
SUB GetDefaultData SHARED
fwCustList$(0) = "Abernathy.Joe"
fwCustNum$(0) = 1
fwFirstName$(0) = "Joe"
fwLastName$(0) = "Abernathy"
fwInitial$(0) = "D"
fwYBirth$(0) = 1961
fwMBirth$(0) = 6
fwDBirth$(0) = 14
fwSSN$(0) = "444-66-8878"
fwCompany$(0) = "First Word"
fwAddr1$(0) = "P.O. Box 66046"
fwAddr2$(0) = ""
fwCity$(0) = "Houston"
fwState$(0) = "TX"
fwZip$(0) = "77266-6046"
fwAreaCode$(0) = 713
fwPhone$(0) = "526-9711"
fwLastContacted$(0) = "12/25/1988"
fwNotel$(0) = "Journalist; has Great...
... Dane; likes girls with glasses."
END SUB

' -----
' Save new or updated info for a customer.
' To add a new customer to the database,
' your user interface must load the arrays
' at index #NumCustomers+1, then pass 0
' for whom% to this procedure. To update
' an existing customer, pass any valid
' customer number.
SUB FWWriteCust(whom%) SHARED
NewFlag% = 0
FileIndex = FileIndex + 1
IF whom% = Nil THEN ' Was value passed?
NewFlag% = 1
NumCustomers = NumCustomers + 1
' Amount offset into database:
CustIndex% = NumCustomers

```

```

ELSE
  CustIndex% = whom%
END IF
FWFile$ = fwLastName$(CustIndex%-1)...
... + "." + fwFirstName$(CustIndex%-1)

\ Truncate if needed for ProDOS:
IF LEN(FWFile$) > 15 THEN
  FWFile$ = LEFT$(FWFile$,15)
END IF

\ Place in correct DIR:
Whom$ = "**/SYSTEM/CUSTOMERS/" + FWFile$
x% = CustIndex% - 1
OPEN Whom$ FOR OUTPUT AS FileIndex
PRINT #FileIndex, STR$(fwCustNum$(x%))
PRINT #FileIndex, fwFirstName$(x%)
PRINT #FileIndex, fwLastName$(x%)
PRINT #FileIndex, fwInitial$(x%)
PRINT #FileIndex, STR$(fwYbirth$(x%))
PRINT #FileIndex, STR$(fwMbirth$(x%))
PRINT #FileIndex, STR$(fwDbirth$(x%))
PRINT #FileIndex, fwSSN$(x%)
PRINT #FileIndex, fwCompany$(x%)
PRINT #FileIndex, fwAddr1$(x%)
PRINT #FileIndex, fwAddr2$(x%)
PRINT #FileIndex, fwCity$(x%)
PRINT #FileIndex, fwState$(x%)
PRINT #FileIndex, fwZip$(x%)
PRINT #FileIndex, STR$(fwAreaCode$(x%))
PRINT #FileIndex, fwPhone$(x%)
PRINT #FileIndex, fwLastContacted$(x%)
PRINT #FileIndex, fwNotel$(x%)
CLOSE #FileIndex

\ Add customer to list if new:
IF NewFlag% THEN
  AddCustList
END IF
FileIndex = FileIndex - 1
END SUB

\ Private procedure - Add customer to list
\ of data file names
SUB AddCustList SHARED
  OPEN "**/SYSTEM/CUSTOMERS/CUST.LIST" FOR...
  ... APPEND AS FileIndex
  PRINT #FileIndex, whom$
  CLOSE #FileIndex
END SUB

\ -----
\ Shut down the data file library, release
\ memory, clean up flags.
SUB FWShutDown SHARED
  FileIndex = FileIndex + 1
  UpdateFWData \ Update master data file

\ Release array storage. Note that if you
\ are exiting the application, you do not
\ need to erase the arrays. If you do wish
\ to erase the arrays, AC/BASIC requires
\ that you make this code a subroutine
\ in the main program:
\ ERASE fwCustNum%
\ ERASE fwCustList$
\ ERASE fwFirstName$
\ ERASE fwLastName$
\ ERASE fwInitial$
\ ERASE fwYBirth%
\ ERASE fwMBirth%
\ ERASE fwDBirth%
\ ERASE fwSSN$

\ ERASE fwCompany$
\ ERASE fwAddr1$
\ ERASE fwAddr2$
\ ERASE fwCity$
\ ERASE fwState$
\ ERASE fwZip$
\ ERASE fwAreaCode%
\ ERASE fwPhone$
\ ERASE fwLastContacted$
\ ERASE fwNotel$
  DataReady = 0
  FileIndex = FileIndex - 1
END SUB

\ -----
\ Remove a customer. which% should be a
\ valid customer number. If it is not, the
\ procedure will return with data unchanged.
SUB FWDeleteCust(which%) SHARED
  FileIndex = FileIndex + 1
  CustIndex% = which%
  FWFile$ = LastName$(CustIndex%-1) + ...
  ... "." + FirstName$(CustIndex%-1)
  IF LEN(FWFile$) > 15 THEN
    FWFile$ = LEFT$(FWFile$,15)
  END IF
  Whom$ = "**/SYSTEM/CUSTOMERS/" + FWFile$

  \ Make sure file exists:
  OPEN Whom$ FOR OUTPUT AS FileIndex
  CLOSE Whom$

  NumCustomers = NumCustomers - 1
  KILL Whom$
  f$ = "**/SYSTEM/CUSTOMERS/FW.TEMP"
  x$ = "**/SYSTEM/CUSTOMERS/CUST.LIST"
  OPEN f$ FOR OUTPUT AS FileIndex
  OPEN x$ FOR INPUT AS FileIndex + 1
  WHILE NOT EOF = FileIndex + 1
    LINE INPUT #FileIndex + 1; z$
    \ Everything except the customer
    \ being deleted:
    IF z$ <> Whom$ THEN
      PRINT #FileIndex, z$
    END IF
  WEND
  CLOSE #FileIndex
  CLOSE #FileIndex + 1
  KILL x$
  NAME f$, x$ \ FW.TEMP -> CUST.LIST
  UpdateFWData \ master data file
  FWShutDown \ Clear cust. arrays
  FWDataSetUp \ Re-init data
  FileIndex = FileIndex - 1
END SUB

\ -----
\ This updates the master data file. Add and
\ remove saved fields as your needs change.
SUB UpdateFWData SHARED
  FileIndex = FileIndex + 1
  OPEN "**/SYSTEM/FW.DATA" FOR OUTPUT...
  ... AS FileIndex
  PRINT #FileIndex, STR$(NumCustomers)
  CLOSE #FileIndex
  FileIndex = FileIndex - 1
END SUB

\ -----
\ End of AC/BASIC Data File Library

```