The Boston Computer Society TI-99/4A User Group Newsletter May 1990

Edited by J. Peter Hoddie

The Fayuh

On May 5 the Fifth Annual BCS TI Fayuh was held in Waltham. This year it was held together with some participation from the Commodore user group, although it was overwhelmingly TI in nature. While attendance wasn't huge, a the local crowd was there in force, and was very enthusiastic. Rave 99 was there showing their standard compliment of fine products, along with a new expansion box. Priced at around $300 it was an amazingly slick unit. I heard rumor that of 10 available for sale, 8 were spoken for by 1 PM. It is slated to ship sometime in June. Bud Mills traveled up to the show again this year, to demonstrate and promote his ever widening line of peripherals. Asgard was represented in part by Mickey Schmidtt who again trekked up from Pittsburgh. Mickey was also selling booklet aimed at helping cassette owners get the most out of their system. The booklet was a compilation of articles she had previously published, reprinted with assistance from Mike Wright. For the first time in my life I had the pleasure of meeting John Willforth - the TI community's famous hardware guru who was also visiting from Pittsburgh. MYARC was represented by Paul Charlton, who was also helping me out at the JP Software table. Also present at the JP table was Mi Kyung Kim, the artist responsible for the GENEVE swan and many other MY-Art pictures, attending her first TI show. Mi Kyung was selling a disk of five new MY-Art pictures she has created. The disk is available for $5 from JP Software, with the profits going to the Red Cross in San Francisco. Mi Kyung also amazed attendees by creating some pretty amazing MY-Art images during the show. Wayne Stith was also there. Wayne spent much of the day demonstrating his TRIAD for the 9640 - and amazing all who saw it. John Birdwell was not there, but his new version of Disk Utilities (currently called Disk One) was well receiving by those who got a chance to see it.

Terrie Masters was there, playing Chainlink at the JP Software table, and otherwise promoting her current interests (I hear rumor of a new publication). Another person there from the west coast was Ken Hamai, UCSD P-System fan, and all around entertaining kinda guy. The prize for furthest distance traveled goes to an Australian man, whose name I missed. Also present was Jeff Guide, head SYSOP of the TI-Net on Delphi, and Jim Horn - Jeff's counterpart on CompuServe. For the first time in awhile, Steve Lamberti of Texaments was at a TI show, promoting TI-Base and its many fine add-on products. The local deals were there too, Ken Gilchrist and Frank Billari. The Connecticut Users Group put together a fine display as always. Jack Sugrue was up and around again, as enthusiastic as ever. Mike Wright managed to dig up a few more treasures in his effort to create the perfect TI shrine. Barry Traver was sorely missed, but family matters came up. Corson Wyman was there, the last weekend before his wedding - the ceremony performed by, who else?, Barry Traver. As always, Corson was most helpful - proving equipment, time, and enthusiasm to kill for. The regular BCS crew was there too - Justin Dowling - who did a great job organizing the whole thing, Donald Mahler - sort of the ultimate BCS volunteer, Tom Ward - one of the most under-appreciated members of the TI community, and many many others.

If you weren't at the show you missed something. There were lots of folks from all over the place. If you had a question, there was someone there who could have answered it. If you had an idea for a program, there was someone there who might have been talked into writing it. If you wanted an informative way to spend a Saturday afternoon, you missed a great informational gathering. (You missing some great Indian food too.) The people were there. The information was there. The products were there. The rest is really in your court.

The show was a good time. I came three thousand miles for it, and I think it was worth it. These TI gatherings are as much about the computer as they are an opportunity to get together with a group of old friends. Most vendors had good things to say about their sales. The crowd was very interested in the products, and asked lots of questions - which makes the show more interesting from my point of view.

If you've got ideas about how to make the show better, you should start talking now. Tell us the things you like, the things you don't like, and what you'd be willing to do to help out next time around. The TI community may not be growing but there is still enough people, enough talent, and enough interest to keep things going if everyone is willing to do just a little bit. (end of sermon - and, by the way, probably the reason I did this newsletter rather than sleep and try to recover from this cold).

Introduction to the UCSD P-System
By Ron Williams
Using Units

This month I am going to show you how to create a P-System unit and also how to add the unit to your system. A unit is a separate compiled section of code that is called by a program or programs. The great thing about a unit is once it is in your library you can call it to be used by any program. So making a unit for getting the system volumes you could use this unit by many programs that need the volume names. All you have to do is put at the beginning of your program a USES statement this will tell the compiler to look for a unit to be linked to the host program. After the unit is called by the program the computer will then search the drives for the unit. Many different procedures or functions can be put in one unit just call them like a regular function or procedure as if the program had them all compiled together. The P-System unit has two main parts the interface section and the implementation section. The interface section declares how the host program communicates with the unit while the implementation section defines how the unit accomplishes its task this is the section that the code for the unit is stored. When you start a unit you declare it like this 'UNIT <UNIT NAME>' you use the statement UNIT and not PROGRAM. After this you start the interface section and this is where you put any functions or procedures to be called by the host program. The implementation section is where the code for the procedures and functions in the interface section is put. A unit can put in the system library or it can be in a library defined by the program LIBRARY.CODE on the P-System utilities disk. You do not even have to put the unit in a library if you use the compiler directive '$U' this directive will tell the compiler where to find the unit. I like to compile a new unit and use it for a while before I add it to a library as I sometimes have my boot disks very full of data, and each time I change a library I have to (K)runch all this information back together. The following unit I put together so you will see how a unit and host program work together. The unit is very simple it just gets the volume names as they are stored in memory. The only information the unit needs is the volume number as they are called by the P-System (4,5,9,10). I have included volume

number 10 for you people lucky enough to have four drives on your system. Here's the code for the unit and host program, Thanks:

```
Unit getvol;
  Interface
    Procedure volume (driveno : integer ; var volout : string);
  Implemention
    Procedure volume;
    Type
      instring = string[12];
      vol = record
        case boolean of
          true : (address : integer);
          false : (ptr : ^instring);
        end;
    Var
      volumes : vol;

    Begin
      case driveno of
        4 : begin
              volumes.address:=14022;
              volout:=volumes.ptr^;
            end;
        5 : begin
              volumes.address:=14034;
              volout:=volumes.ptr^;
            end;
        9 : begin
              volumes.address:=14082;
              volout:=volumes.ptr^;
            end;
       10 : begin
              volumes.address:=14094;
              volout:=volumes.ptr^;
            end;
      end;
    end;
Begin
End.
Program testunit;
  USES (*$U #9:unitvol.code*) getvol;
  var
    volin : integer;
    outvol : string[12];
Begin
  outvol:='            ';
  page(output);
  Writeln('This program tests the');
  Writeln('getvol unit this unit ');
  Writeln('will get the volume names');
  Writeln('from memory locations and');
  Writeln('can be called by giving a');
  Writeln('drive number then printing');
  Writeln('out the string with the');
  Writeln('volume number in it');
  writeln;
  write('Enter volume number=>');
  readln(volin);
  volume(volin,outvol);
  Writeln;
  Writeln('Volume in #',volin,' is-',outvol);
end.
```

Introduction to the UCSD P-System
By Ron Williams
Configuring your System

So far I have not covered how to make life a little easier using the P-System. I know that you would like to set your system up the way you need it for your use. I will now go into where files should be for your system to boot up the way you want it to. First you should know that some files must be on volume #4 or you will not be able to make the system boot correctly. One of the most important files on volume #4 is SYSTEM.PASCAL this is the operating system file you need this file on volume #4. This file is a supplemental file to the operating system file on volume #14. The file SYSTEM.SYNTAX should be on this volume as well this is where the compiler error codes are kept if you don't have this file available you will just get numbers for compiler errors. This can be a problem I don't like to look in books and manuals to find the meanings to the error numbers as this can take a lot of time. If you have a file called SYSTEM.CHARAC at boot up you will not get TI's default character set. This is good because you will not have true lower case characters with TI's character set. A file on volume #4 that tells the system where the library files are located is called USERLIB.TEXT. This is a simple text file created with the editor that has the file names of all the libraries in the system and which volumes that they are located on. If you have this file on volume #4 you can then put the libraries on any volume to save space on volume #4. Just type in the file names just as you would to execute a file, volume number or name along with the file name.

The compiler file, assembler file, and filer files can all be put on volume #5 to save space on volume #4 for writing programs. Some of the most used files from the utilities disk I also have on this volume. I will print out at the end of this article file listings from the disks I use to boot my system to help clear up any questions on where files should be put.

The file I use to set up my system at boot up is called SYSTEM.STARTUP and is located on volume #4. This file is the first file the system executes after booting. In this file I have a program that sets my printer to 'PIO' and polls the drives to see if the disks I have are DSDD or SSSD or any other setting. Polling the drives is needed on my system as I have a MYARC disk drive controller and it will not read DSDD disks correctly in the P-System unless the drives are read first for disk formats. The part of the program that sets my printer to 'PIO' is just a modified version of the program MODRS232 on the P-System utilities disk. The program then uses the chain command to go directly to the filer program and starts the date command. So right after the system boots I am in the filer program and the computer is waiting for me to set the correct date. Before I did this many times I forgot to set the date and so my files were not dated correctly. The chain command used is on page 51 in the compiler manual. The program that uses this command must be compiled using the unit COMMANDIO. At the top of my program I put USES {$U #5:COMMANDIO.CODE} COMMANDIO; the unit is on volume #5 in my system. At the end of my program just before 'END.' I put the following statements:

```
MYOPTIONS:=CONCAT('*SYSTEM.FILER. PI="D',' ',' "');
CHAIN(MYOPTIONS);
```

The variable myoptions is of string data type. The command will execute the file SYSTEM.FILER after the program SYSTEM.STARTUP stops. The chain command will also do simple commands that you would normally type in after the program chained to starts that is how you can start the date command. Look at the above statements and I think you can see what I mean. I will now show you listings of my disks to

help you along farther. A few of the files on volume #5 like TEXTFILE.CODE, P-DIR2.CODE and PASCDV80.CODE are programs that I wrote they have nothing to do with the booting of the system. Also you will find a few files that were taken from the P-System utilities disk these programs are ones which I use quite often.

```
DISK4:
SYSTEM.PASCAL       6 20-Oct-81    10 Code
SYSTEM.STARTUP      3 29-Jan-89    16 Code
SYSTEM.SYNTAX      14 12-Jan-82    19 Text
SYSTEM.LIBRARY     43 17-May-87    33 Data
SYSTEM.CHARAC       2 14-Jan-82    76 Data
SPECIALLIB.CODE    15 24-Jan-88    78 Code
USERLIB.TEXT        4 31-Jan-88    93 Text
< UNUSED >        263              97
7/7 files, 263 unused, 263 in largest

DISK5:
SYSTEM.FILER       34 13-Mar-81    10 Code
SYSTEM.COMPILER    99 27-Apr-82    44 Code
SCREENOPS.CODE     12  2-Jun-81   143 Code
SYSTEM.EDITOR      45  6-Nov-81   155 Code
SYSTEM.ASSMBLER    49 10-Apr-81   200 Code
SYSTEM.LINKER      28  4-Apr-81   249 Code
9900.OPCODES        3 20-Dec-78   277 Data
9900.ERRORS         7 23-Sep-80   280 Data
COMMANDIO.CODE      8  2-Jun-81   287 Code
DFORMAT.CODE        7  9-Mar-87   295 Code
LIBRARY.CODE       13 19-Oct-81   302 Code
SETLTYPE.CODE       7 10-Feb-82   315 Code
PASCDV80.CODE       7 24-Jan-88   322 Code
TEXTFILE.CODE      10 13-Aug-88   329 Code
P-DIR2.CODE        16 25-Jun-88   339 Code
< UNUSED >          5             355
15/15 files, 5 unused, 5 in largest
```

That's it for this month I hope that this information will help you set up your system just how you want it.

ForTI Card on a 9640?
From Delphi's Message Base

The following information is taken from the message base on Delphi. It contains information about how the ForTI twelve voice music card's hardware appears on the 9640. I do not know of anyone who is successfully using a ForTI on the 9640. Can anyone out there help?

Like the speech synthesizer, the ForTI card is located at page >BC. It is not fully decoded, and therefore could respond at pages >3C, >7C, and >FC as well. Here is the ForTI address bus mapping:

Address bit	Source	Selects	Address	Sound Chips Selected
AME.A	NC	X	>8400	ALL
AMD.A	NC	X	>8402	2-4
AMC.A	Mapper	1	>8404	1,3,4
AMB.A	''	1	>8406	3,4
AMA.A	''	1	>8408	1,2,4
A0	''	1	>840A	2,4
A1	''	0	>840C	1,4
A2	''	0	>840E	4

A3	9995	0	>8410	1-3
A4	''	0	>8412	2,3
A5	''	1	>8414	1,3
A6	''	X	>8416	3
A7	''	X	>8418	1,2
A8	''	X	>841A	2
A9	''	X	>841C	1
A10	''	X	>841E	None
A11	''	CE4	active LOW sound chip 4 enable	
A12	''	CE3	'' '' '' '' 3 ''	
A13	''	CE2	'' '' '' '' 2 ''	
A14	''	CE1	'' '' '' '' 1 ''	
A15	''	0		

On the 99/4A, writing to >8400 will load data into all ForTI sound chips as well as the console sound chip. Writing to any ForTI sound chip will also write to the console sound chip, I think. The only way to write to the console sound chip without writing to the ForTI sound chips is by using address >841E. Note that the states of A6-A10 do not matter, such that >8400, >8420, >8440, >8460, etc., are all equivalent ways of accessing ALL sound chips, and so forth.

Jeff White

P.S.: When I put >BC at >8004 in the GPL mapper, the system stops.

MY-Art File Formats
By J. Peter Hoddie

Recently I received a request from an individual about the MY-Art file format. Since this information is not entirely obvious, and is not widely available (but was once published on CompuServe).

There are two different MY-Art file types: low-res and hi-res. Both files save in nearly the same format - a form of run length encoding. The first two bytes of the file are a flag indicating whether or not the file is hi-res or lo-res. These are not always accurate. Earlier version of MY-Art sometimes saved them incorrectly. In an ideal world, the first word is ">0FFF" to indicate hi-res and the first word is ">FF00" to indicate lo-res.

In hi-res mode the user can only access 16 colors, but can choose them from a palette of 256 colors. The hi-res mode header includes this color information, the lo-res header does not. The hi-res mode color header information lists 16 colors, each entry is one word long. The default color header is as follows (taken from the MacFlix source code).

```
       DATA >0000,>0200,>3000
       DATA >3200,>0003,>0203,>3003
       DATA >3203,>7204,>0700,>7000
       DATA >7700,>0007,>0707,>7007
       DATA >7707
```

In hi-res mode, the horizontal scan lines data then follows. It is encoded with the color code in the high nybble and the number of pixels in that color is stored in the remaining three nybbles. The count should not be greater that >0200 (512 decimal) since that is the maximum number of bits on one horizontal line in hi-res mode. There are 212 horizontal lines in total. A blank line appears as >F200 (color is white, and 512 bytes).

In lo-res mode, the horizontal scan line data is stored in a similar format. In lo-res mode however, there are only 256 pixels across the screen and there are 256 colors to choose from. Therefore, the color is stored in the high byte and the pixel count is stored in the low byte. The pixel count is one based, and the number 256 is stored as a zero (since it can't run into the next byte). Therefore, a blank line in this mode would appear as >0000. Again there are 212 horizontal lines.

Both of these file formats are stored as DIS/FIX 128 data. If the data does not completely fill a sector, the remaining space in the sector is ignored. There is no end of data mark. The picture data is considered to have ended when 212 horizontal scan lines have been read.

If you are writing a program to import or display MY-Art pictures, you should consider totaling the number of dots in each horizontal scan line to make sure they total 256 or 512 (depending on the mode you are in). If you find a line that goes over this you should alert the user to the error. This helps to avoid crashes caused by corrupted files.

As noted above, sometimes the header work indicating whether the file is in hi-res or lo-res mode is incorrect. There is a way to determine nearly for certain what type the file is. Ignore the header word and assume that the file is hi-res mode. This means that the data immediately following the header word should be a line of horizontal data. Start parsing the data (but not displaying it) and see if it totals 512 pixels of data, or it runs over. If the data you are parsing is hi-res data, then it will total 512 pixels. If it is lo-res you will be totaling the color data, which most certainly will not exactly equal 512 pixels. Adding this sort of test requires a little extra effort, but it spares your users the pain of trying to display files in the wrong way.