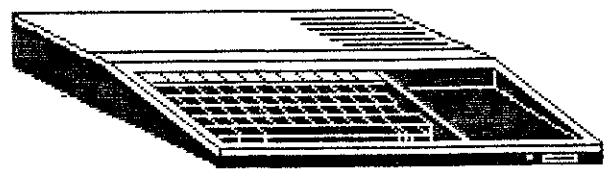
A publication of the TI 99/4 and 9640
Ogden users group inc.

OGDEN TI 99-4A USERS GROUP



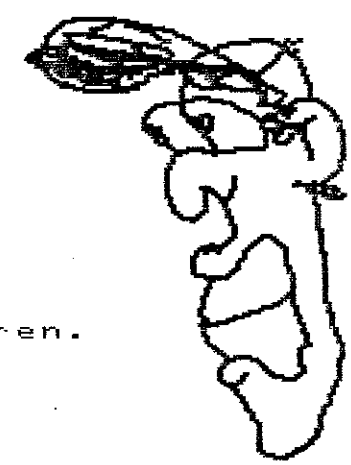
THE CLUB THAT REFUSES TO SAY GOODBYE

SEPTEMBER 1989

OUR FIRST SATURDAY MEETING HAS BEEN CHANGED THIS MONTH TO THE 09th ONLY!

YES!

EUGENE, You should go to school; like the other children.



WELCOME!

EDWARD E ISLER
TO OUR GROUP.

TI-BASE - From INSCEBOT
TUTORIAL 9.2.1 By Martin Smoley
NorthCoast 99'ers - May 20, 1989

I hate to have to make a correction already, but I must. Because I have the update chips in my TI 99/4 Impact Printer, I was able to set my line spacing for graphics in 216ths of an inch. The original TI Impact Printer is restricted to line spacing in 72nds of an inch. I can't go into it at this time, so if your graphic prints out in separate segments, check your printer manual for the proper codes.

The CF below is the last item I mentioned last month. It demonstrates a couple new tricks. One: Instead of using local space, we are displaying fields directly to specific screen locations. Two: Each time DSKNAP1 runs, it will display the current Running TOTal on disks, which is kept in slot 3. This total will be updated by the CF INVUPDT. Everything else is fairly simple. You are asked for the number of disks requested, or the number to be shipped. The answer is stored in ANS and if greater that zero DSKPRL1 is run. DSKPRL1 is the CF all our Graphics people will be interested in. It is very simple in it's straight through nature, but complicated in it's use of data and commands, which are accessed throughout the TI-Base system. NOTE: In an effort to help you make some sense out of this information I will reference past tutorials with the use of brackets. If you encounter a [8.1] for example, this means that a particular item, or idea, was covered in tutorial 8.1.

```
*                              DSKNAP1
     CLEAR
   WRITE 5,9,"     NAME FOUND"
   WRITE 9,3,"  Exp. Date    ",XP
   WRITE 12,3,FN
   WRITE 12,18,LN
   WRITE 14,3,SA
   WRITE 16,3,CT
   WRITE 16,22,ST
   WRITE 16,25,ZP
WRITE 20,1," Disks in stock =  ",3.RTOT
  WRITE 22,1," Disks requested = "
 READ 22,22,ANS
WRITE 22,1,"                                "
IF ANS > 0
  DO DSK2.DSKPRL1
ENDIF
 CLEAR
RETURN
*
* DSKNAP1     Save as  DSKNAP1/C
* *******            03/31/89
```

The first thing done in DSKPRL1 is a check to see if our stock of disks has fallen below 50. If it has, the CF NOTE1 is run. NOTE1, which I listed on the next page, puts the message, "ORDER MORE DISKS", in the upper right corner of the screen. The message is stored in the Db MSRET, which we have opened in slot #4, [9.1.4]. The first PRINT statement starts the interesting stuff. PRINT 4.CD2 means print whatever is in field CD2, in the Db we have opened in slot 4. Once again we are using MSRET, but this time the field is designated X-type, [9.1.2]. The field contains 1B3324, and a bunch of zeros. TI8 will interpret this as Hex code because of the X designation, and send it to the printer. The printer will see it as the Hex Code to change the line spacing to 36/216ths of an inch, [9.1.3].

```
*                              DSKPRL1
* Copyright Martin A. Smoley 1989
*
  IF 3.RTOT < 50
    DO DSK2.NOTE1
  ENDIF
    PRINT 4.CD2,(e),(E),4.TO,(Drft)
  REPLACE TEMP1 WITH "                 ";
     ! "  Exp. Date " ! XP
    PRINT (E),TEMP1
  REPLACE TEMP1 WITH TRIM(FN) ! " ";
           ! MI ! " " ! LN
    PRINT TEMP1
    PRINT SA
  REPLACE TEMP1 WITH TRIM(CT) ! " ";
           ! ST ! ". " ! ZP
    PRINT TEMP1
    PRINT (CR),(LF),4.CD1
*
    SELECT 5
    FIND "OHIO"
    PRINT (E),(e),4.FR,(Drft)
    PRINT (CR),(LF),(E)
    PRINT 5.GR1,(CR),5.GR1,4.NAME
    PRINT 5.GR2,(CR),5.GR2,(CR),(LF)
    PRINT 5.GR3,(CR),5.GR3,4.STREET
    PRINT 5.GR4,(CR),5.GR4,(CR),(LF)
    PRINT 5.GR5,(CR),5.GR5,4.CTSTZP
    PRINT (CR),(LF),(CR),(LF),(Drft)
*
    FIND "DISK"
   PRINT 4.CD1,(E),(e),4.CTN,(CR),(e),;
     4.CTN
    PRINT (CR),(LF),(E)
    PRINT 5.GR1,(CR),5.GR1,4.CD
    PRINT 5.GR2,(CR),5.GR2,(CR),(LF)
    PRINT 5.GR3,(CR),5.GR3,4.DNB
    PRINT 5.GR4,(CR),5.GR4,(CR),(LF)
    PRINT 5.GR5,(CR),5.GR5,4.DNX
    PRINT (CR),(LF),(CR),(LF),(Drft)
    SELECT 1
RETURN
*
* DSKPRL1     Save as  DSKPRL1/C
* *******            04/01/89
```

Continued Next Page.

TI-BASE - From INSCEBOT
TUTORIAL 9.2.2 By Martin Smoley
NorthCoast 99'ers - May 20, 1989

36/216ths inch amounts to normal line spacing. The zeros are used for fill and will have no affect on the printer. NOTE: zeros are the only value that will have no affect on the printer. (e) is my printer Control Code for one line Enlarged Print, and (E) for Emphasized print, [8.1]. All this is to print TO: in Enlarged and Emphasized mode, at normal line spacing. The (Drft) then changes the print type back to Draft Mode. All the work you just went through is to demonstrate the ability to use data, built in printer controls, and controls of your own from special data fields in the same statement. The REPLACE statement is standard. We are placing some words and the persons expiration date into TEMP1. The print line changes the printer to Emphasized, (E), and prints everything we placed in TEMP1. The rest of this is standard, down to the PRINT (CR),(LF),4.CD1. In the past I initalized a BLANK and then printed it for blank lines. (CR) and (LF) will do the same job with no need for memory space. 4.CD1 sets the line spacing to 24/216ths inch, for the graphics. "AH the Graphics." SELECT 5 means make slot 5 TIB's main slot for a while. This is necessary for the FIND function. Our graphics are located in the DB named GRF1, in slot 5, [9.1.2, 9.1.3]. It is sorted on the field GRFNM which allows us to find the graphic data by it name, ie; OHIO. This technique would work quickly for a larger number of graphics and the names could be menu selectable. After OHIO is found we print From:, in the same manner as above, feed one line and reset Emphasized. The (E) will not affect the graphic, but will affect my name in 4.NAME. We then print 5.GR1, the first line of graphic, do a Carriage return with no line feed and print 5.GR1 over again. This is a Double Struck Graphic. Then we print my name, 4.NAME, in normal Emphasized mode. The next line prints the next part of the graphic with no text after it. Here is an important note. The graphic data I have set up in GRF1 has no CR or LF attached. TIB does not hang CR, LF on the end of X type fields. Actually it is better that way. However, this means that you must send a CR and LF after each graphic. The LF is the one that really fires the graphic print. This need not be done if normal data is sent after the graphic, because it will have it's own CR, LF on the end. If you compare the different graphic print lines, you'll see what I mean. You should also go over [9.1.3] a couple times. The last line in this group prints a couple blank lines to get to the next blank label. This is something you must remember when working with 15/16" labels. It is exactly one inch from the top line on a label, to where you print the top line on the next label. If you are working with 216ths inch increments, you must move 216/216ths to get to the next label. During the time we are printing the graphic label, we are feeding lines of 24/216ths each. This means we can feed 9 lines total to get to the next label, 9X24=216. If it doesn't work out that neatly, you can do a one time line feed at the end of the label to make up the difference. This value can be placed in an X-type field and used wherever you need it. I know this stuff sounds complicated and maybe even trivial, but if you're short even one or two 216ths, the printing on your label will slowly creep up until it's off the label. I've had this happen many times and you have probably had the same problem.

The next label prints out in the same manner, but the data is changed. Because of the FIND "DISK" statement, when GR1, GR2, etc. are printed, we now get the disk graphic instead of OHIO. The SELECT 1 statement points TIB back at slot 1 before the RETURN to the previous CF. It is then possible to find another name and do the whole thing again. One thing I'd like to cover is INVUPDT. This small CF is in charge of keeping track of the disk supply and who got what on which day. The first step is to SELECT 2 and APPEND a BLANK. This means we are working with SLSREC, which is in slot 2, and we have added some space to it so we can save some data to that space. The next three lines save the ID number, the present date and the quantity of disks shipped, from the members data named on the mailing label, to the SLSREC Database. This data could be used later to find out how many disks were shipped to whom, on what dates.

That's about it for this month. The LIMA meeting date is approaching rapidly and I haven't prepared my demo yet. Along with pages 9.2.1 and 9.2.2, I have included a TI KEY/CHARACTER CHART by JIM SWEDLW, I hope there will be room for it in the newsletter. I have been using this chart constantly to convert ASCII values to HEX. If you are working with the X-type fields, the only Hex you should need are >0 to >FF, or 0 to 255. This chart is for XBasic, but it really helps with everything we have discussed in the last couple tutorials.

NOTE: I intend to do a few more things with this graphics series, and the retrieval of information from multiple databases simultainously. Keep an eye on this column for the next few months.

```
* Copyright Martin A. Smoley 1989
*                                  NOTE1
WRITE 2,28,4.MS1
WRITE 4,28,4.MS2
WRITE 6,28,4.MS3
WRITE 8,28,4.MS4
WRITE 10,28,4.MS1
RETURN
* Print a message from the DataBase
* in slot 4.   This is CF  NOTE1/C
* DB in  slot 4  should be  MSRET
*
**********************************

* Copyright Martin A. Smoley 1989
*
*                                INVUPDT
SELECT 2
APPEND BLANK
REPLACE ID WITH 1.ID
REPLACE SDT WITH .DATE.
REPLACE QTS WITH ANS
SELECT 1
REPLACE 3.RTOT WITH 3.RTOT - ANS
REPLACE 3.LDT WITH .DATE.
RETURN
*
* INVUPDT  Save as INVUPDT/C
* *******  Ver. 2.01 04/01/89
```

Continued Next Month.

TechTalk

-By Mike Maksimik

Some of you may have followed TI's developments in the time that the 99/4A was at it's childhood. All sorts of plans, marvels, new things for the home computer that "was ahead of it's time." There were several peripherals developed by TI but were only released in tiny quantities, mostly to the TI employees that got the pick of the crop. Some of these never made it to the production lines, but only a few prototypes survived.

The modem card, which essentially was a Novation Cat 300 baud modem, was placed on a peripheral card, and a DSR ROM was given it to control very low-level functions, such as modem-to-vdp RAM interrupt routine, powerup routine, etc. It would work with a command module, like TE II just as the disk manager module works with the low-level routines in the disk controller to perform the DOS functions. Only a very few of these survived.

Another little known card was the IEEE 488 bus controller card. It contained the TMS9914 GPIB (general purpose interface bus) that allowed the lab and mechanical equipment that used GPIB to interface to the TI. One could access the GPIB like a file device. This same standard is found in unexpected places. Any of you have a commodore 64? The communications bus used to connect it's ring-style bus of peripherals is a modified GPIB, one of commodore's own design. The SCSI interface (small computer systems interface) is essentially a multi-GPIB, allowing very fast buffered serial transfer between storage devices. SCSI also has interrupt lines to alert the host that data is waiting to be read or written. The VCR controller, a $500.00 range peripheral, along with support software, was introduced as a means to combine video from a VCR and the video from a TI. The card would control playback, hold, framing, and other functions. Digital Research created a similar product to control videodiscs that attached to an apple or a commodore 64, although much later than TI's development.

The debugger card, a little known device, was in existence when the 99/4A was born. In fact, it's design can be rooted to the support hardware in the 990 minicomputer series. Essentially, the TMS9900 is a minicomputer on a chip. The editor/assembler GROM was a virtual image of the DX10 assembler used on the 990 minicomputer. Some directives one would only find on a minicomputer exist in the editor/assembler package, but were dormant in the 99/4A. The debugger board was designed to bring the 99/4A closer to a minicomputer's environment. The DEBUG program, included with the editor/assembler package, has several features that cannot be used without this piece of hardware. In fact, the editor/assembler looks as if it was taken direct from a 990 itself. The only added features were the GROM utilities, sucha VMBW, DSRLNK, LOADER, etc. that didn't support the features that a 990 could handle. It's too bad that TI wishes to keep the plans for this card on ice, it would be a dream to program with. It allowed multiple breakpoints by using the XOP 3 opcode, which would allow you to step your program through and look for errors or miscalculations. Although we can do this through software, the debugger board used a hardware approach.

The design of this board, and what it contained, are up for grabs. If anybody knows, I'd appreciate you sharing with the rest of us. Send me a letter. Still another rare peripheral was the GROM library peripheral. It essentially was a super-widget that could access ALL of the GROM in the cartridges. This would be handy for TI BASIC, since TI BASIC searches external GROM for subprograms. TI extended BASIC does this too, but doesn't search DSR ROM when a program is running. Modules like TE II, personal record keeping, and extended BASIC could all be plugged in and the CALL routines could be accessible to BASIC. BASIC could use the commands it wished to whatever, and all you had to do is plug your favorite "flavor" modules into the library peripheral to get the necessary language expansion. Imagine a GROM cartridge giving advanced graphics to TI BASIC, another for print spooling, still another for expansion memory control. Others for high speed cassette routines, etc. so the language could expand by adding cartridges. It's the same technique used with the peripherals: the computer never becomes obsolete, because it automatically responds to any new device attached. This is true of the library peripheral. This is another device I would LOVE to see.

Some of us have the HEX-BUS controller. In the days of the 99/2, the CC40, and the 99/8, the hex-bus controller was introduced for the 99/4A to allow compatibility with these devices. Essentially, they were designed like the commodore 64's peripheral system, where a slow serial transfer was appropriate for the hex-bus devices, a disk drive wouldn't be feasible. So TI never considered the HEX-BUS disk drive. The Wafertape drive, the CAT modem, the RS232/parallel interface, and the 4-color printer, were all developed. All were battery operated and could fit in a briefcase, as did the CC40. For the 99/4A, it was an inexpensive means to expand.The hex-bus controller was a small device containing a DSR ROM that controlled the I/O drivers which "spoke" to the hex-bus peripherals. Since the main use was for the CC40, it wasn't pushed for the 99/4A. The 99/8 could also rely on the PE BOX for it's devices. It had it's own special FLEX CABLE card, which used some special control lines to expand it's own capabilities.

Since the 99/8 used a TMS9995, the same as the GENEVE, it could use the extra 3 address lines in the PE BOX, giving a total address space of 2 to the 19th power, or 512 k of directly addressable memory. Since some of these banks were probably switched, the address space grew to a total of 4096 k, which is sufficient for MOST of my needs. The speed of this processor was greater, and it's throughput was even greater, but more on that later. Some other control lines were used, some to indicate a 9900 or a 9995 present in the system, some to allow multi-level interrupts, still others to initiate HOLD sequences, which are found on the mainframes, and large multi-user systems as a way to deal with wasteful processing, and interrupt idling. TI had a HARD DISK controller in the plans, probably MYARC's, but the technical data I have is 1982.

I own a rare card. Some of you may remember a company called A/D electronics, out of Sacramento, California. They produced a control card which allowed sampling of environmental data through an 8-bit analog-to-digital controller. This device allowed hookups of many items, such as temperature probes, light transducers, etc. and was mainly used as a scientific device. Some possible uses included home control, because it also contained a real-time battery backed clock. Plus, there were separate digital inputs and outputs, for switches and relays, respectively. My main use for the A/D card, FIRST ADE, is a mouse. The RADIO SHACK color mouse contains two potentiometers turned by a rolling motion of the mouse. The potentiometers, when interfaced with the ADC0809 chip, (two channels, x and y) gives me mouse control with TI ARTIST. I wrote the DSR myself, and have been using this device for about a year and a half. The MBP clock card is a similar device, although it does not contain a digital input or output array. The ADE card, however, could also switch external relays, or sample data on 16 lines (8 in, 8 out). If timing was correct, an 8-bit parallel interface was possible. I still use this card, and the clock is handy for keeping my p-system master disk up-to date.

The FORTi music card was a device which allowed one to produce sound on not one but 4 extra TMS9919 sound generators. By arranging the frequencies on the 12 music channels available, different waveforms were possible. Now, with the FORTi, sounds even a c-64 owner could envy were possible. And, there were 4 percussion channels independent of each other. I can imagine "AXEL-F" running on this card!!

And of course, we all know of the more common peripherals, the triple tech, the disk controllers, the 32k cards, te rs232 cards. Even these make our computers sophisticated enough to meet TI's long dead expectations. I also own the p-code card, and another article is devoted to THAT!

I mentioned the TMS9995 earlier. Just what exactly is a pipeline microprocessor? Well, the 9995 is not only fast, but it has a distinct advantage over others in it's class, even the intel 80386. Those processors rely on expanded address lines and increased instructions to increase throughput. There was a deeper approach, one that TI envisioned in the 9995. A pipeline microprocessor is one that incorporates special hardware that allows it to have more than one part of the microprocessor running at the same time. These CONCURRENT functions provide that while one instruction is being decoded inside the chip, another is being fetched from memory. Still another is being executed after it has been decoded. At best, with top-down code, and very little jumps, the microprocessor can achieve a throughput 3 times, or more, depending on the level of pipelining, over a regular processor running at that speed. For example, if we put test code into a 9995 and a 9900 running at 12 MHZ, the worst case is that the two run even. But the 9995 can pipeline, and with the pre-fetch and post-store the 9995 can LOOK like it's running 16, 20, or even 24 MHZ. And with the reduced instruction set in the control ROM, the 9995 has a distinct advantage over an 80386, it's MUCH cheaper to produce. The control ROM is a hard-wired design, while the 80386 has to be programmed externally. It is an easy device to interface to a memory system, and with no-wait state static RAM, the memory-9995 combination (up to 4 megabytes) can be phenomenal.

Currently, I am working on a software project. It's a new DOS for the TI, somewhat reminiscent of COMMAND DOS that ryte data released some years ago. However, there is no image file required because the DOS I have resides in a E/A supercart, and the utilities that it needs are extracted from the E/A GROM--that way, I can restore the lower memory expansion to a defined state very quickly without reading from a disk drive. The DOS is completely self contained, and will provide a choice for you on the master title screen. I am a college student, doing projects to complete my final years of undergraduate study in computer science. This project was inspired by a need for a better operating environment for the TI as well as a need for me to see if it could be done. Well, I have succeeded! The DOS uses the DSRLNK utility to attach to the low level device drivers. It gives you the familiar A> DOS prompt, and will mimic DOS to a degree, but with one delightful exception--the DOS is being written by me, and I can have it do whatever I want it to! I will no longer be a slave to incomplete DOS commands or ambiguous and useless syntax, often the product of overpaid software developers. The commands are clear and precise, and the DOS is very short, only about 5k at this writing. Since most of the DOS is already present in our machines, in places like the E/A GROM, the disk controller ROM, the RS232 interrupt routine--all of these put together with the right glue can make a great DOS, and all I did was to provide the necessary glue for the parts, and it works! It has a batch file load and execute, D/F 80 loader (compressed/uncompressed), program file loader, dos utilities (FORMAT, COPY, RENAME, DELETE, ASSIGN) and screen control commands (WAIT, BEEP, CLS, GOTOXY, PRINT, ECHO ON/OFF) and "smart" control keys, as well as a 255 character input queue for type-ahead. Many of the commands are internal, and they reside only in the supercart. Other commands can be created from object code, which you can create from any one of the compiling languages, or the assembler (i prefer the assembler) and by simply typing the name of the file at the command prompt, the file will be loaded and executed.

I hope to have some sort of language compiler for DOS, such as a basic/pascal compiler, to facilitate creation of programs and utilities. My plans include a file transfer utility (terminal emulator), windowing, an 80-column editor, and multiprogramming. If for no other reason, then to gain experience and to enjoy doing it on my $49.99 TI99/4A. Of course, I wouldn't dream of charging anyone for this DOS, and I've had some interesting suggestions for names. "F-DOS" by our own editor, BOB DEMETER, for FROGMAN-DOS, since my "other" hobby is SCUBA DIVING, "XIOS" for eXtended Input Output System, and whatever...I am using version 1.24, which is relatively complete. I would just like to add the bells and whistles, plus write a manual on it's use.

Now for some more TechTalk. If you are confused as to why computers like the c-64 and the apple all have DOS commands built in...well, the designers of those computers anticipated a disk system, and available to most users, so the operating system and BASIC language all had the DOS commands either in the disk unit itself, or in a disk BASIC which loaded in on powerup. Since TI did things a little differently, they preferred to make DOS a separate thing, with a disk manager module to handle disk tests and formatting. It seemed a little annoying that in order to rename a file from BASIC, you had to either load the program and save it under another name, or if it was a DATA file, you had to OPEN it and read all of the data, then re-save the data to disk under another OPENed file name. This could be terribly inconvenient to users, but consider what the others have...the c-64 must send all of it's DOS commands through a command channel, and the disk drive will run itself. It essentially is another computer, a 6502 based one, to be exact, that only accepts commands from a serial line and performs all of the disk commands. Imagine.. a computer so STUPID that you need TWO computers to run any disk software...and you would be paying for TWO computers also. Commodore doesn't tell the average users that they are essentially using TWO computers instead of one.

Apple computers are also based on the 6502 series of microprocessors. Apple used an old method of running it's computers...just write a DOS and put it on disk, and when the computer is powered up, the DOS is loaded. Funny thing, though. Although Apple boasts of 64k of RAM, much of that is used to hold the resident DOS, and BASIC. If you want to load a program which needs the space allocated by DOS, you are out of luck, since your program might make DOS calls to perform disk functions. And if DOS were overwritten, then when your program is finished, it must go back and load it all over again. And 6502 is not exactly the processor I would waste terribly expensive memory on, since it has a very limited instruction set, and things I take for granted now, like memory-to-memory word moves, multiplication, division, and subroutine branching would be terrible to implement on an apple of commodore 64. I just don't know how they have survived this long...

Our little TI, on the other hand, has a wonderful method for handling new devices. The GROM header, present on all ROM in the expansion box, and all command modules, is the link between the unknown and the known. It allows us to plug in new devices at any time in the future, and the operating system will immediately recognize the device, as if it were there from the beginning. This is what will keep our TI computers alive. The method of access is very similar

to the IBM pc method. Each peripheral card has a certain address in the serial addressing fields. The operating system can turn on a card singly, look at what occupies a pre-defined memory area (>4000 to >5FFF for us) and can determine if the device exists. With the IBM, certain logical names are assigned to a physical device address, such as COM1:, TTY:, A:, LPT1:, and so on, and can be changed according to the user's wishes. This requires a small modification to DOS to accommodate the new device, and from then on, a new sub-version to dos is created. If the device is removed, an error will be issued since DOS can no longer locate the installed device.

The GROM header in the TI provides a standard table for finding a device quickly and efficiently. All of the devices use a pre-decoded 8k block of memory, and 8k is plenty for most devices. Since we are not limited to 64k of total address space (via memory paging in the MYARC or HORIZON ram cards), larger programs may occupy that memory and give our TI's a greater running capability. The IBM uses a segment register that is pre-decoded to page in banks of memory, which is essentially the same way the HRD or MYARC does it, so memory expansion is no problem. The safe area in the TI is the first ROM bank, which is the invaluable interrupt routine and powerup routines. the SUPERCART is the only save RAM alternative for a kernel or DOS, since it is battery backed and it remembers all the changes you have made to DOS. In the CRU, the only area you could use for your own bit-twiddling is the >400 to >1000 area, which is not decoded presently and could be wired to something (I will let you imagine that). It would not be a difficult task to interface an IBM card to the TI, provided you had the correct cross-wiring, and a ROM to control the new device. A few of us in the chicago users group will attempt this. The price of IBM cards is falling like a rock, and I don't see any interfacing pitfalls.

Using IMAGE Statements

The following program is an example of how to use the IMAGE statement in TI XBasic. The program itself is just an example and may need modification or other changes in order to prove useful. However a careful study of the listing may reveal some techinques that will be helpful in your own programs.

TI in many of its tutorials, presumed a more in depth knowledge of programming than many users have. The descriptions and examples they offer are often hard to follow and because of this many important uses of XBasic have gone un-used. We hope this brief article and programming example will help you to better understand the IMAGE statement.

The print-out has been reduced in size to fit our newsletter format, but when the program is run it fills the 8 x 1/2" size of your paper.

Lines 120 through 170 specify where the money figures are to be printed on the page. Line 180 prints a single underline for sub-totals and line 190 a double underline for the grand-total. Line 200 accumulates the various sub-totals and line 210 adds the sub-totals to arrive at the grand total.

Lines 220-360 names the various categories and their dollar amount and print them according the IMAGE statement specified. The Category (CAT) headings were positioned according to line 110.

Of course, any and all of the headings, categories, dollar amounts and totals can be changed according to your needs. We hope this little program will help to shed a little more light on how to use the IMAGE statements.

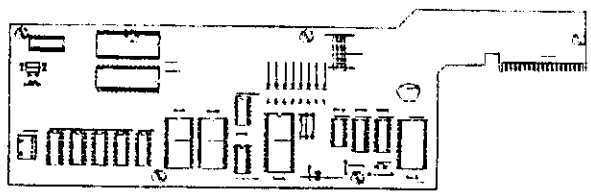
```
100 OPEN #1:"PIO"
110 PRINT #1:TAB(33);"CAT.I"
;TAB(45);"CAT.II";TAB(57);"C
AT.III"
120 IMAGE $$$$$$$$$$$$$$$$$$
$$$$$$$$ $$$$$$.##
130 IMAGE $$$$$$$$$$$$$$$$$$
$$$$$$$$  $$$$.##
140 IMAGE $$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$ $$$$$
$.##
150 IMAGE $$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$  $$$$
$.##
160 IMAGE $$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$ $$$$$.##
170 IMAGE $$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$ $$$$$.##
180 SUL$="---------"
190 DUL$="========"
200 A=34.95 :: B=567.33 :: C
=39.95 :: D=444.99 :: E=87.6
5 :: F=263.28 :: G=9873.96
210 H=A+B :: I=C+D :: J=E+F+
G :: K=H+I+J
220 PRINT #1,USING 120:"EXPE
NSES",A
230 PRINT #1,USING 130:"TAXE
S",B
240 PRINT #1,USING 140:"MILE
AGE",C
250 PRINT #1,USING 150:"REPO
RTS",D
260 PRINT #1,USING 160:"FARE
S",E
270 PRINT #1,USING 170:"CAR-
RENTAL",F
280 PRINT #1,USING 170:"MAJO
R-PURCHASE",G
290 PRINT #1,USING 130:"SUB-
TOTALS",SUL$
300 PRINT #1,USING 130:"",H
310 PRINT #1,USING 150:"",SU
L$
320 PRINT #1,USING 150:"",I
330 PRINT #1,USING 170:"",SU
L$
340 PRINT #1,USING 170:"",J
350 PRINT #1,USING 170:"",DU
L$
360 PRINT #1,USING 160:"GRAN
D-TOTAL",K
```

	CAT.I	CAT.II	CAT.III
EXPENSES	$ 34.95		
TAXES	567.33		
MILEAGE		$ 39.95	
REPORTS		444.99	
FARES			$ 87.65
CAR-RENTAL			263.28
MAJOR-PURCHASE			9873.96
SUB-TOTALS	602.28	484.94	10224.89
GRAND-TOTAL			$11312.11

ANNOUNCING THE "INTERNAL BOARD"

The "Internal Board" AKA the "Zenoboard" is now in production. The PC board is being manufactured by one of the best quality manufacturers around. As promised, this board will allow you to build 32K memory, a clock circuit and add your extended basic and speech synthesizer to the interior of your console. Orders are now being accepted at a cost of $17.50. Documentation will consist of approximately 8 pages of schematics, builders notes, parts list, software for the clock and parts placement overlay.

Specifications:



* 32K STATIC RAM
* battery-backed clock
* speech synthesizer
* extended basic
* 3 additional, switched grom sockets
* any circuit configuration can be used
* requires no additional power
* move your extended basic and speech synthesizer inside
* eliminates nearly all lock-ups due to extended basic cartridge
* in addition to soldering components connectors, only 12 additional wire conections have to be made to build a complete board
* switches lights may be added to turn off any or all circuitry
* grom reset switch
* compatible with all other known hardware/software
* circuit designs have been tested

Orders and technical questions/comments can be directed to:
Eric Zeno 414 Highland Rd., Pgh, PA, USA 15235 (412)371-4779
The boards are being manufactured in quantities of 100. Don't delay and get caught waiting for the subsequent manufacturing runs!

Small modifications must be made to the plastic on the inside of the console.
Hand soldering skills are recommended.
Please allow sufficient time for delivery.
Overseas shipping may require additional invoicing.
Sorry no C.O.D., US currency only!

SHIPPING		
	USA	OTHER
FIRST PC BOARD	$2.50	$8.00
EACH ADDITIONAL BOARD	$ .50	$6.75

	QTY.	AMOUNT
PC BOARD $17.50 X		
SHIPPING SEE CHART		
8 PAGE DOCUMENTS $1.00 X		
-TOTAL-		

SHIPPING ADDRESS:

NAME:

STREET:

CITY: STATE: ZIP:

ERIC ZENO
414 HIGHLAND RD.
PGH. PA. 15235

HOW TO LOAD A PROGRAM: Stephan Meyers

Chicago Times

It has come to my attention that many novice users are having a lot of trouble loading and running programs, particularly when given a disk full of programs. In fact, I am getting rather sick of phone calls from people saying "how do you boot up..." and such. Thus, I will endeavor to show you exactly what to do to figure out how to load a program. This article assumes that you are already versed with the care and feeding of disks: file management, etc. You should keep a write protect tab on the disk until you are sure of what you are doing. On with the show!

First and most important: CATALOG THE DISK. This applies to any disk you get. Catalog it first opportunity' That way, if anything is wrong with it, you can get it fixed. You should catalog to a printer, but a cursory peek on the screen will do if you have a good memory. If this is a disk you have bought, there may be a copy protection on it, which could make it uncatalougable. In this case, your documentation should tell you what to do.

If there is a PROGRAM file named LOAD, you should probably put the disk in drive one and go into Extended Basic. Many programmers set programs up to load quickly and easily in Extended Basic, and it is best to take advantage of this when available.

The next thing you'll want to do is categorize the files on the disk. I do this in my head, but it's best to write it down. Catalog the disk to printer, or copy the directory by hand. Divide the files up into categories: PROGRAM, D/V 80 (Display Variable 80), D/F 80 (Display Fixed 80), and Wierd. Wierd is Internal files, and wierd length Display files (D/V 20, D/F 69)

Next, look at all the D/V 80 files on the disk. Look especially for a file like *READ/ME or DOC1 or INSTRUCT or some such thing. You can use the new 3) READ D/V 80 FILE option of Disk Manager 1000, but I recommend using TI-Writer. That way, you can catalog the disk and load up any D/V 80 file you find, and skip around reading it at will. However, do not write the file back to disk. Just load it, read it, and load the next D/V 80 file or quit. Don't ever change anything on a disk until you know what you are doing!! A D/V 80 file may not be documentation: it could be data, or even program if the author is doing something wierd. If you don't understand what the file is, ignore it and load another one. You'll probably hit instuctions eventually, if there are any. Another thing most people ignore is option 4 of the E/A editor: print. This will print a D/V 80 or D/F 80 file from disk to printer. At does not load the editor, so it is the quickest way to get a hardcopy of a disk file. Give the disk filename and printer filename when asked, and just sit back and let it print.

Files that are what I call Wierd type are almost certain to be data, although they could be program files if there is a special loader. By the way, I speak of program files as opposed to PROGRAM files. You will soon know the difference.

Program files, the kind that show up as PROGRAM in a disk directory, can be many things. They are most likely to be Basic or Extended Basic, or Program File Assembly. They can also b e data, most commonly with drawing programs. Look at the PROGRAM files to see if there is a pattern of a list of similar names with only the last character differing by one. For example, FAST-TERM and FAST-TERN, DM1 and DM2, or BIG-PROG, BIG-PROH, and BIG-PROI. The program file loader expects a program too big for one file to be in multiple files with the last character increased by one (M,N; 1,2; or G,H,I). To load this series of files, use Option 5 of the Editor/Assembler module, or any of the public domain option 5 program loaders floating around, such as XB-OPT5 by Barry Boone. Give the first filename, DSK1.BIG-PROG, and the file, along with all of its subsequent files will be loaded and run. Remember, if you change the filename of one, you have to change them all: if you make BIG-PROG into PROGRAM, the other files must be PROGRAN and PROGRAO in the proper order. Usually, but not always, all of the files except the last one will be 33 sectors. This corresponds with 8K of memory, the size of the buffer used by the program image saver usually used to create assembly language program files. A small assembly language program can be in one file, which loads the same way as multiple files. Try each program file in option 5, paying special attention to the first one in a list as detailed above: I.E. You really don't have to bother with trying PROGRAN and PROGRAO, but you should pay special attention to PROGRAM, since it is the first in a list. If this does not work, the file is probably a Basic language file. Try loading it in Extended Basic. If that doesn't work, ignore it, it's probably data.

Ogdens 99'er User Group
Mail Address: 1396 LINCOLN Ave. Apt.B Ogden Utah 84404

GROUP OFFICERS

President: JimBuck 773-2552
Vice President: Harold Bingham 394-6382
Secretary-Treasurer: Richard Scott 776-2551
Librarian: Harold Hilburn 773-0622
Asst. Librarian: Mel Bragg 393-9605
Newsletter Editor: Mel Bragg 393-9605
Associate Editor: Harold Hilburn 773-0622

SEPTEMBER 1989 NEWSLETTER

OUR NEXT MEETINGS ARE:

SATURDAY: SEPTEMBER 09 TIME: 0900 hrs.
TUESDAY: SEPTEMBER 19 TIME: 1900 hrs.
We will be meeting in the CIVIL AIR PATROL building at the OGDEN MUNICIPAL AIRPORT AIRPORT ROAD.

OGDEN TI USERS GROUP
1396 LINCOLN APT #B
OGDEN, UTAH 84404