THE GUILFORD 99'ER NEWSLETTER

VOL.5 NO.4 APRIL 1988

Janice Snyder, President — Bob Carmany, Vice President
Mack Jones, Secretary/Treasurer — Herman Geschwind, Program Library
BBS: (919)274-5760(OPUS) — ROS (919)-621-2623

The Guilford 99'er Users' Group Newsletter is free to dues paying members (One copy per family, please). Dues are $12.00 per family, per year. Send check to 3202 Canterbury Dr., Greensboro, NC 27408. The Software Library is for dues paying members only. (George von Seth, Editor)

OUR NEXT MEETING

DATE: April 5, 1988. TIME: 7:30 PM PLACE: Glenwood Recreation Center 2010 S. Chapman Street.

Program for this meeting will be using a disk sector editor. We will explore how to alter and customize programs at the most basic level ---by directly editing the assembly language code on the disk. Two of the most popular programs will be demonstrated --- DISKO and DISK UTILITIES by John Birdwell. In addition, there will be a text-to-speech demo using just the XB cartridge for unlimited speech capability.

ATRAX TRACKS

By Bob Carmany

So far in this series we have looked at the CONFIG program to reconfigure your FUNNELWEB system and also at the very first menu that presents itself. If you have taken the time to get FUNNELWEB configured the way that you want, you are ahead of the game. If not, this month we are going to look at the TI-Writer functions and the options that are presented when you choose this option from the initial screen.

If you choose 'TI-Wr' as your initial choice, you are presented with another series of options: 1) Editor, 2) Formatter, 3)DM1000, 4) Modem, 5) Data-Base, 6) Utility, and 7) Users List. Let's examine them one at a time.

EDITOR selects the TI-Writer Editor with all of the capabilities that you are familiar with. There are, however, some VERY welcome enhancements. First of all, you will notice that there is now a scale at the bottom of the screen that makes it easier to plan your character positions. If you type across the screen, you will notice an "end of line" marker in the form of a "beep" five characters before the end of the line is reached. Another enhancement to aid you in planning your text position. If you read through the documentation that came with F'WEB, you will find that the "end of line" warning can be changed or eliminated entirely if you wish with a minimum of effort.

(S)how(D)irectory now presents you with a series of options that were not available previously. That is because while F'WEB was loading, a file called QD was also booted giving you this enhanced menu. After choosing the drive number that you wish, a listing of programs is displayed on the screen. All of the prompts that you have available are also displayed at the

bottom of the screen. If there is more than one page of entries, pressing <SHIFT> will page back and <CTRL> will page forward. Pressing "=" will give you a program check. That is, is will cause the program-type files to be parsed and F'WEB will tell you if they are XB programs or E/A program-image files. You can also mark a D/V 80 file for loading simply by pressing the appropriate number. The filename is then placed into the 'mailbox' for loading with (L)oad(F)ile after you re-enter the Editor. No more typos when loading files!! You can also (V)iew a D/V 80 file on the screen without disturbing whatever text you have in memory or print a directory with (P)Dir. (D)elF will delete a file from the disk (you won't see any change until you exercise 'SD' again. You can use (O)ldF to restore a filename into the 'mailbox' after you have marked a file for viewing, for example. All in all, quite an improvement over the old 'SD' function in TI-Writer.

There are a couple more options that I find to be of value from time to time. <CTRL .> will change uppercase to lowercase and <CTRL ;> will do just the opposite. If you have even forgot to release the alpha-lock key and typed in a line of text, you can readily see what value this option has --- no re-typing.

Of course, you can configure your printer specs to whatever your system calls for by using the CONFIG program. This can be done for both the Editor and the Formatter. You would expect all of these enhancements to take up some of the text buffer space but, amazingly enough, it didn't. So you have the same amount of text buffer space that you had with TI-Writer.

FORMATTER loads the TI-Writer Formatter that you are all familiar with. It has all of the "old" standby options that you would expect --- and some more enhancements! You can permanently configure your printer option to match your printer specifications. By pressing <FCTN-7> you can exercise the 'SD' function to look at any drive that you wish (file marking is not operative). In fact, only those options that are available are displayed at the bottom of the screen when the disk directory is displayed. A nice touch --- you never need remember prompts!!

If you are working on a file with the Editor and then switch to the Formatter to print the file out, the filename is retained in the "mailbox" and is displayed as the default filename in the Formatter. This is just another convenience that TI should have done when they had a clean slate to work with.

To set you default drive designation for the F'WEB system, all you have to do is follow the instructions and prompts when you run CONFIG for the first time.

That brings us to option 3 ---DM1000. This is Vn. 3.5 with some enhancements added. Now when you configure your printer, you can use 3 character printer codes and the results can be sent to any disk drive instead of 'DSK1' as a hard-coded choice. Oh, <CTRL-3> from the title screen allows you to set your printer options for DM1000. 'File Utilities' now contains a choice to reload F'WEB and asks for a drive designation for the TI-Writer and E/A halves of the program (you can seperate them on different drives if you wish). Exercising this option when you are through will flawlessly reload F'WEB.

The next option is 'Modem'. This provides for loading a terminal program from drive 1 with the filenames of MD, ME, etc. If you wish, you can edit this entry on your work disk with a sector editor and insert the name of your favorite telecommunications program (10 characters max). To install MASSTRANSFER, for example, simply re-name the files MASS and MAST to MD and ME respectively. TELCO works much the same way by re-naming the files TELCO, TELCP, and TELCQ to MD, ME, and MF. The only restriction is that the F'WEB file EA and the re-named modem files must be in drive 1 when the 'modem option is chosen.

The next option is 'Data-Base'. Again, this option can be edited with a sector editor and your 10-character favorite used by name. The files that this option loads sjhould be named DB, DC, etc. PR-BASE loads very nicely when re-named and installed this way. The restriction is the same for the previous option --- that is, EA must be on the disk in drive 1 when this option is chosen. Of course, the easiest way to do it is put all of these programs on a single DSSD disk along with the F'WEB files and keep a seperate disk for the other options. Or you can use several SSSD disks and install EA on each of them and switch disks before you choose the group of options that you want to use.

The next option is 'Utility'. This option can be changed with CONFIG and loads a file called SP from drive 1. The option name can be changed with CONFIG as well to whatever you choose to load in this space. On my system, I load SPELLCHECK. The conversion is really simple. I just added EA to my SPELLCHECK disk and rename the UTIL1 file to SP. It loads flawlessly! Ther are also some instructions in the documentation for making SPELLCHECK return to the title screen when it is finished instead of unceremoniously crashing.

The last option is 'Users List'. Again, using CONFIG, you can install any A/L program that you wish in the spaces available (you can't modify the last entry, though). From this menu, you can load A/L object code (Load and Run), A/L program image files, and GPL environment program image files! There is even an option 'Next UL' which can be used to access another menu by editing the UL file and renaming it UM. Theoretically, you could string an infinite number of Users List files together and load virtually every A/L program that you have!! Incidently, it will also load both TI-Forth and Wycove Forth with the proper entries (TI-Forth needs the file XB4TMLD).

Well, that takes us through the 'TI-WR' option from the main menu. As you can see, there are numerous enhancements and improvements that have been added to what is probably the best word processing package anywhere. If you have any questions about F'WEB or how to configure it to suit your particular system, drop me a line at our UG address.

Next month, we are going to look at the second option on the main menu -- the Editor/Assembler and the menu screen that comes with it including the A/L loader package that will allow you you to load just about any imagineable assembly language program.

XB TUTORIAL

By Tony McGovern

It's time once again to get back to the regular Tutorial material, continuing with the ways and means of scrunching program length. As I remarked before, it's a subject I'm not completely comfortable talking about because, while this series has been devoted to better XBasic programming, most things you can do to scrunch Basic programs make them less readable by ordinary mortals, given reasonable programming skill in the first place. The other reason for my reluctance is that this kind of discussion tends to degenerate into a collection of unrelated items, yet another set of "Tips", when I really want this series to be a gentle but systematic look at the workings of the machine and its language(s).

Anyway let's start at the small end of things and work up to the larger scale. Last time we looked at the space taken by simple variables. The most obvious thing is to keep variable names short. I don't recommend this until late in the piece because it is such a cheap and obvious way of gaining bytes that you might as well have the help of descriptive variable names until you are absolutely desperate for bytes. Absolute desperation has not occurred until you have had several rounds of byte saving already. The shortest variable name has only one letter character, but TI Basics also officially allow "@" (shift-2) and "_" (fctn-U) as variable names. It has to be a fairly long SUBprogram before you need more than 26 simple numeric variables but it can happen. On this console there are 3 other single characters which can be used as variable names. Experiment to find if they exist on your machine. The nagging problem is that they are not documented.

There is another way to use variable names to shorten a program. Remember from last time that a one digit numeric constant is treated as a string and takes 3 bytes, while a single letter variable takes only 1 byte. If a particular numeric value occurs frequently in a SUBprogram, 0 or 1 being common examples, then it may be worth the overhead, 14 bytes plus the defining statement, for a new variable of that value if you can then save 2 bytes on numerous occasions. A frequently used longer numeric constant, as might occur in CHAR or SPRITE manipulations, yields more bytes each time. It is a matter of doing careful book-keeping and byte counting in each SUBprogram. Once you start down this track be alert for further gains -- if you have defined G-7 and F-5 then it saves a byte to write G*F instead of 35. If you can reuse an already defined variable name then the investment is paid back faster, but this requires keeping very careful track of program flow. Go back to the example of a Key/Joystick routine in an earlier Tutorial and see if you can shorten it by reducing the number of variables used.

Replacement of numbers by variables has precedents in other languages. In TI-Forth the numbers 0,1,2,3 are not treated directly as numbers but are defined words in the language.

There is another little way that cunning entry of characters can shorten programs. This is in the entry of graphics characters with ASCII values above 127 in the upper color groups of XB by writing strings with DISPLAY AT instead of H & VCHAR CALLs. Characters in this range can be entered in strings in program statements by use of the CRTL key, rather than by using the CHR$ function. It does tend to make the program incomprehensible as these echo as blanks to the screen. They will appear with their defined shapes if the line is called up for editing after RUNning the program. These codes are also used as XB tokens and can only be used within strings. I should add in passing that I am in total agreement with the TI designers' choice not to allow abbreviated (direct token) entry of Basic keywords. If you want that sort of thing you should be back on your Sinclair or Commodore, and you probably don't believe in relocatable object files either,

The use of arrays to represent small collections of numbers needs detailed working out. The gains from less variable table overhead and simplified parameter passing to SUBprograms have to be balanced against the extra bytes needed for each program reference. Let the program logic be your initial guide.

This idea of using fewer bytes to represent quantities leads on to the larger subject of data compaction. One byte can carry 256 different values, and one third to one half of those can be conveniently entered from the keyboard. It's sheer overkill to use an 8 byte floating point number to represent just a few values, or even just a logical (Boolean) variable which really needs only one bit. Some languages compact Boolean variables as bits in a word or words. The CRU single bit bus of the TMS-9900 provides an ideal mechanism for bit storage and testing, but as in so many other areas the 99/4a hardware does not do justice to its CPU. The later TMS-9995 in fact has a little on-board CRU memory for just this purpose.

Opportunities for data compaction are limited in XB both because of the structure of the language (it has only character strings, floating point numerics and arrays of these as data types) and the convoluted, slow way it is implemented via GROMs and VDP memory. Any scheme for coding or compacting needs computation to pack and unpack the data. At the machine code level the tradeoffs between memory use and speed are different from those in Basic, especially TI-99 Basics, because Basic is so much slower. In my experience the use of string variables to compact data in active parts of a program is almost always doomed to failure because of slow string handling by XB and pauses for garbage collection. Data compaction can be useful though in setting up initial graphics designs or for music data. There are only so many different notes, in pitch length and volume used in any given short musical piece, and since each note takes time to play and is handled by the machine on an interrupt driven basis, this time can be used to do the computations needed to unravel the data for the next note.

Let's have a look at the graphics screen example. Suppose that in setting up a game screen, either one of two

characters, maybe the same pattern in two different color groups, has to be written to 20 locations in various parts of the screen. The simplest way is a whole succession of CALL HCHARs - assuming the display is not suited to generation with DISPLAY AtS - and that's the way you will find it done in many programs (just like long lists of CALL SOUNDs). What is totally unforgivable is to find incompetent magazine or commercial programs with inefficient coding that force inconveniences like CALL FILES(1) on the user.

1000 CALL HCHAR(23,12,105) 1010 CALL HCHAR etc etc This takes over 600 bytes. How can it be shortened ? One way, a bit of a dead end in this example, is to use multi-statement lines. This would be shorter by 30 bytes or so, and marginally faster. The real improvement is to eliminate the repetition of CALL HCHAR - remember CALL is cheap but HCHAR is expensive - by using a loop and DATA statements.

```
1000 FOR I=1 TO 20 :: READ A
,B,C :: CALL HCHAR(A,B,C)::
NEXT I
1010 DATA 23,12,105, etc etc
```

Now all but one of those HCHARs have gone. The price paid is loop and DATA execution overhead and the increased possibilities for clerical errors since the DATA items have been divorced from their proper context. At this stage you may be feeling very pleased with yourself, but then you find that to add another feature to your program you need more space. Now is the time to reflect seriously on data compression. A column index for HCHAR can only have the values 1 to 32 and rows 1 to 24. One of these values can be expressed by 1 byte with possibilities to burn. Say you use 1 byte for each row or column value then. Expressing the bytes efficiently as DATA is the next problem - there are a few bytes of overhead for each item in a DATA list, and DATA lists of a lot of short items are notorious for causing a "line too long" error. So let's pack them in a single string and use SEG$ to unpack them, with ASC to turn a ASCII character back to a value for HCHAR. A minor problem is that characters 1 to 32 can't be entered directly in XB, so just use characters starting with "A" and subtract 64. The opposite problem may occur with the string for the character values if upper graphics sets are being used. Then just use lower values and add a correction. So now the code might look like

```
1000 READ A$,B$,C$ :: FOR I=
1 TO 20 :: CALL HCHAR(ASC(SE
G$(A$,I,1))-64,ASC(SEG$(B$,I
1))-64,ASC(SEG$(C$,I,1)+32):
: NEXT I
1010 DATA "W... ","L... ","I.
```

You could further pack the data into a single string and modify the SEG$ statements accordingly, but it might not be worth it. Remember now that the problem posed involved writing only two different characters and work out how you could compact things still further for this limited case. This example is based on one of methods that was used to squeeze TXB into console memory. An extreme example of data compression comes when the data is regular enough that it can be generated by a formula or procedure. This is something that has to be worked out in each case.

The use of loops as in the examples above applies in other situations, particularly in CHAR definitions. XB allows the use of multiple arguments in CHAR, COLOR, SPRITE and suchlike SUBprograms. This is better and faster than using individual SUBprogram CALLs for each item in the list. The real dilemma comes when you try to use a loop to compact the program further. Critical parts of the program may be slowed down unacceptably so that you may find yourself using compact slow code in some parts of a program and longer but faster forms elsewhere. Just in passing I should remind you to null out on exit from a SUBprogram, any string variables not required to keep their value till the next CALL. This particularly applies to string variables used for READ, INPUT, PRINT etc operations involving long strings. Remember that it is the length of a program while RUNning that really counts.

Time to sign off for this issue now. Next Tutorial will continue with more aspects of byte saving.

FORTH TUTORIAL

By Lutz Winkler

You have determined which of the editors suits you and found a display color you like. They could be entered from the keyboard each time FORTH is booted. But there is a better method: Let the disk do it for you! To begin with we'll use the simple - and later on a more elegant - way. (If you haven't made up an overlay yet, better do it now, else editing isn't going to be easy. Programming in FORTH is done by editing SCREENS and the various editing functions are made a lot easier if you can refer to the overlay.)

So boot your FORTH disk again and when the MENU shows up, enter either -EDITOR or -64SUPPORT. Now get out your manual and go to Appendix I (Contents of the Disk) and look at SCREEN 3. This is the one that gives you the first inkling that something is going on by displaying "BOOTING". So you get an idea of the way FORTH works, let's scan its content before going on :

Line 0: The parenthesis ( ) act like a REM in Basic, so we see that it is called the Welcome Screen. GOTOXY is like

DISPLAY AT, note the coordinates 0 0 preceding it.

Line 1: Forget the BASE->R for now, but let's do someting with HEX. From your keyboard enter

HEX 83C2 DECIMAL .

Don't forget the period, actually a FORTH WORD called DOT. (Look up each word in the GLOSSARY!) What did you get ? -31806 is correct. In plain English line 1 states: Switch to BASE 16, put >10 (16) on the stack, and C! (C-STORE, see page 17, Glossary) it at 83C2. This is how FORTH does the CALL LOAD for FUNCT-Quit Off. (You have seen that one before!)

Line 2: DECIMAL returns us to Base 10, ignore the (84 LOAD), 20 LOAD loads SCREEN 20 (look at scr # 20 and you'll see that it's the menu which appears at boot-time. 16 SYSTEM is CALL CLEAR (more about System Calls later) and finally MENU displays the menu. Take a moment to digest this, as it gives some idea as to how FORTH works. The command 20 LOAD booted scr # 20 at which time a new FORTH WORD was compiled (see scr 20, line 1). MENU is now part of the DICTIONARY. Anytime MENU is invoked, FORTH looks it up and executes it. Try it, enter MENU. You get the menu and 'ok'. If you enter something FORTH can't find you'll see a '?', sometimes followed by an error message (see Appendix H).

OK, back to the Welcome Screen. But now let's put it on display. Enter 3 EDIT and watch it come up. Skip to line 4 and note that here we have the menu words defined, i.e. : -EDITOR 34 LOAD ; etc. The first word after a ':' is the new word being added to the Forth's dictionary. Any words that follow must already be in the dictionary, otherwise the word being defined can not be compiled. The definition ends with ';'. Move the cursor to line 12 and enter -EDITOR (or, if that's your choice, -64SUPPORT). Now - if you chose the regular edi- tor - you can type the number from your SEE experiment followed by 7 VWTR. Now hit FUNCT-9(ESCAPE) to get out of the edit mode. Your additions to screen 3 are at this time only in a buffer, not on the disk. In order to record them on the disk you must enter FLUSH.

Remember: Every time you EDIT a SCREEN you must FLUSH, otherwise all your efforts will be for nought.

So let's check if your edit was successful. Enter COLD. (If you are using the 64 col. editor enter TEXT COLD.) This word is like NEW in XBasic, except you don't have to do anything else, FORTH will re-boot. (It'll take longer now because you are booting the editor also.)

Now let's recap:

You have 'edited' SCREEN 3 so it boots your editor and sets up the screen color for you. This was done while in the EDIT mode. You have also worked in the 'interactive' mode when you defined the word SEE to determine your color choice. In this mode you can try out your definitions before you use them in a program. You'll find this to be tremendously helpful because unlike BASIC there is no need to RUN the whole program to find out what happens.

Having worked my way into TI-FORTH the hard way, I will leave you with a few suggestions which I feel will be helpful:

As you encounter a new word look it up in the Glossary (Appen. D of the manual) to see what it is supposed to do. Mark the chapters and appendices in your TI-FORTH manual for easier access to them. You'll be using it frequently because - even though it may not seem so at first - it DOES contain a lot of information.

Get a FORTH book, preferrably Leo Brodie's STARTING FORTH. It is sold in many bookstores/software houses. The manual (Appendix C) explains the differences between fig-FORTH, which Brodie uses, and TI's implementation of it.

Though it may read like Greek, scan through the manual. As we go along you might just remember having seen something that rings a bell. (Finding it again may be something else!)

AUTHOR'S NOTE:

Since these tutorials were written, a new edition of STARTING FORTH has come out. In it, Brodie uses Forth-83 rather than fig-FORTH. If at all possible try to get a copy of the old book, otherwise Appnedix C of the TI Forth manual will be meaningless and may lead to added confusion.

MOVIES?

By Jack Sughrue

GOOD OL' DAYS

Reviewing books and software for computers is a lot like reviewing movies. Some of that old stuff was great. CITIZEN KANE, WHITE HEAT, the original INVASION OF THE BODY SNATCHERS. All great stuff. But if you watch them, say, with a teenager; someone who is used to the the brilliant colors, stereophonic sound, incredible special effects of STAR WARS, it is hard to convince them that these films, for their day and for the people who recall those earlier film days, are magnificent. Who can forget Jennifer Jones, the Gypsy dancer riddled with bullet holes, crawling and scratching her way to the top of a cliff only to die as she grasped the hand of the evil-doing Gregory Peck in DUEL IN THE SUN? The youth today scoffs at such nobility, such caring. They split their sides at the word-of-honor approach to humor of the Wodehouse movies and books.

What has happened to the world of yore?

Nothing.

It still lives (maybe with an illusory cherry on top) in our memories. What has seemed to make it less than what it was is our own sophistication. Once we know the how and why and where and what and once we've "been there" there seems no purpose in exploring what we already know. Or is there?

This edition of TEXTWARE/SOFTWARE/ELSEWHERE travels to both worlds: the reasonably innocent TI world of Reston Publishing Co. to the high-powered sophistication of a relative newcomer, Asgard Software.

First the innocence (or seeming innocence) of a book by Claire Bailey Passantino called ITTY BITTY BYTES OF SPACE FOR THE TI-99/4A COMPUTER ($6.95, Reston Computer Group, 11480 Sunset Hills Road, Reston, VA, 22090).

It is one of a series of three: MATILDA THE COMPUTER CAT and SCHOOL DAYS. (I haven't seen the other two.)

What kept me from buying this book is the title. Just as I can't stand grownups who talk baby talk, so I can't stand books whose titles are in baby talk.

But, eventually, I succumbed and bought the book, only because there were no new TI books out for so long that I needed a fix.

I'm glad I bought it.

It's a 5X7 book with a ring binding for nice, flat layout. The print is dark and even. The listings are a little larger than the text for nice, easy typing. And, for the most part, the listings are by 10s. The illustrations by Nancy Gurganus are perfect. They go with the text. They are delightful. And, because they are line drawings, are suitable for coloring. (I always keep my colored pencils handy for such activity.)

Because Ms.Passantino is a computer teacher and a parent, she was able to put together a perfect little book for beginners and some beyond- beginners.

I just paused for a half hour while I coffeed up a bit and typed in a few more of these programs (There are 20.), and I find them to be rather cute.

Both the structure and function of this little book are ideal for kids and kid-like adults. Each even-numbered page starts off with the Main Ideas of the program (which is complete on the opposite odd-numbered page.) The ideas include the new things the program will introduce (data statements, CHR$, variables, loops, and so on). Then comes a line-by-line explanation/tutorial that is truly the clearest, most explicit I have ever come across my desk. There isn't an adult BASIC programmer who wouldn't find this interesting. Finally, there are suggestions and a bunch of "Watch out for" items.

The programs could be typed in by youngsters within a sitting. They are not frustrating, and, yet, they DO something.

And I thought about how far I have come in computing these past few years and how I could fairly easily have created some of these programs and, therefore, what good is a book like this to me?

But.

I didn't create these programs.

I did learn some things from this book.

My students at school LOVED it. They all had remarkable success with typing the programs in and playing them.

And they liked the theme. It's just about the only TI book that has a real theme. Space. Everything in the book relates to that theme. This appealed to the kids. What also appealed was the fact that some programs were set up to join together. Not Extended Merge, but to join other lines to make the programs do something else.

This really turned out to be a sleeper for me. There are lots and lots of kids and unsophisticated (computerwise) adults who would love this book and learn from it and be very comfortable with the TI as a result of it. For the rest of us maybe we have kids or grandkids or friends who are ready but need encouragement. Good book.

(I'll review MATILDA THE CAT and SCHOOL DAYS if I am able to locate copies of them.)

This simple little program below from ITTY BITTY is easy for the kids to type in and it is fun for them to figure out how much they weigh on diffent planets. It is also very easy to render slick with a little dash of color and sound.

```
100 REM FROM ITTY BITTY BYTE S OF SPACE
110 REM BY C\B\PASSANTINO
120 REM JS/P\S
130 CALL CLEAR
140 PRINT "WHAT IS YOUR EART H WEIGHT?"
150 INPUT W
160 CALL CLEAR
170 PRINT "WEIGHT ON EARTH:" ;W
180 PRINT "WEIGHT ON OTHER P LANETS"
190 PRINT "_____ __ _____ _ _____ "
200 PRINT : : :
210 PRINT "PLANET","WEIGHT"
220 PRINT "______","______"
230 PRINT
240 PRINT "MERCURY",\37!W
250 PRINT "VENUS",\876!W
```

```
260 PRINT "MARS",\381!W
270 PRINT "JUPITER",2\637!W
280 PRINT "SATURN",1\151!W
290 PRINT "URANUS",\79!W
300 PRINT "NEPTUNE",1\12!W
310 PRINT "PLUTO",\025!W: : :
```

Meanwhile, lets leap to the sophisticated disk-only stuff.

There is a company (Asgard Software, P.O.Box 10306, Rockville, MD, 20850) which has leaped into TI software in a huge way.

Asgard's catalog is the most eclectic one you'll ever come across, and their products are just about the least expensive on the market for extraordinary quality.

They also distribute the best (I MEAN the best!) quality public domain music, games, and utilites programs. All of these and more I will discuss in future columns.

Right now I'd like to tell you about the most extraordinary disk cataloguer I have ever used.

Why I needed another disk cataloguer I wasn't sure. I had four, each reasonably good. All fine when I had a dozen disks of files. Some even okay when I had two dozen disks.

However, with the amount of work I do on TI Writer and Multiplan and my data bases, I find that my disk collection is growing by leaps and blastoffs. Add to that collection the stuff I use with my fifth-grade class, my home applications items, my precious games, my graphics programs, and all my other non-defined items, including the batch of disks of programs I'm always half-way through writing. And! You will get the picture.

No disk cataloguing cartridge or disk or tape I own is sufficient for my ever-growing needs.

In talking to a few TI friends I discovered they had the same problem. For the serious TIer there was no good disk data base. You notice I said "was".

Because there is.

It's a superb cataloguing disk called, appropriately enough, DISK DATA BASE. It does everything. And does it with such ease a manual is not even necessary.

It makes no difference if your library contains 5,000 files (or more, of course). DDB handles it all.

Nothing is forced on you. Do you want your files sorted? It does by disk name or file name. Not bad, eh? But what if you don't want them sorted? Unlike the other four (very limited) cataloguing programs I own, DDB will accept the disks as entered. A very handy feature which I've used already on a few occasions.

Oh, yes. What if you have half-a-million files and you only want to print up (or screen view) just a few sections (say, the ones with your small dog-grooming business on them)? No sweat. DDB lets you catalog into blocks of 250 (thus removing all limits to the total number of files providing only that you have enough data disks). Take out the block or blocks you want, look at them, print them up, sing to them on the housetop. It makes no difference. DDB accepts it all.

It there more to this program?

Well, for starters, there is a very helpful on-line dictionary of terms you can call up at will.

The screening (all extremely fast menu-driven easy-to-understand screens) and sorting is done by superquick assembly language.

You can convert other disk-catalogued files created by other programs into DDB.

If you're just starting? Why not start with DDB. It's the only program of its type that will grow with you.

DDB requires Extended BASIC, 32K, disk drive with controller. A second drive and printer are sure helpful.

(As an aside: my other disk cataloguing programs cost me a total of $169.85. This is terrible. I could have bought three more TIs for that and had money left over to buy DDB.)

The Disk Data Base from Asgard is $14.95. It comes with a program disk and 2 or 3 database disks. Ask for their free catalogue. It's loaded with other inexpensive, intriguing, and intelligent programs.

TEXT-TO-SPEECH

By Bob Carmany

Some years ago, TI offered the then newly-produced Speech Synthesizer as a "free speech" promotion with the purchase of a selected set of software packages. If I recall, you had to send in the package ends and they sent you back a Speech Synthesizer. The price on this "miraculous" piece of equipment dropped until now almost everyone has one attached to their console. It works fine for cartridges with built-in speech but there is little documentation for the independent use of the Speech Synthesizer. That leaves us at the present state of affairs!

This column is going to (hopefully) provide some information about using the Speech Synthesizer for text-to-speech and full utilization of the built-in vocabulary.

Besides the game cartridges that have built-in speech capabilities, there are three other cartridges that can access the speech capabilities of the Speech Synthesizer. The Speech Editor cartridge was one that was produced briefly and, for all

practical purposes, is non-existant. So, we are going to concern ourselves with the two other cartridges that allow for direct speech access. The TE II cartridge is the first of these. It allows text-to-speech and other speech facilities as well. Let's start out by looking at it and what you can do with it.

To understand the TE II cartridge, we need to findd out how the Speech Synthesizer generates the speech patterns that we hear.

The Speech Synthesizer processes speech by using the phonetic pronunciation of each syllable in the word. These phonetic "bits" are known as allophones. There are a total of 127 of these allophones that comprise the entire speech vocabulary. They follow a set of rules that govern how each letter or combination of letters is to be pronounced. Within the Speech Synthesizer, the entire resident vocabulary is stored as a series of allophones at a specific location. The TE II cartridge contains a listing of the 127 allophones that are the same as those used to create the resident vocabulary and are used for the unlimited text-to-speech capability. The interesting thing is that either the Speech Synthesizer or the TE II cartridge can be accessed to produce speech.

There is just one small problem with using allophones to produce speech phonetically. Not every word that you want to have spoken follows the "rules" for producing phonetic speech. As a result, you will have to mis-spell words from time to time to have the Speech Synthesizer pronounce them correctly. But, this does lead to another facet of text-to-speech that has not been fully explored. By phonetically spelling words, you can produce speech in foreign languages. For example, try the German equivalent of "Merry Christmas" --- phonetically spelled --- "FROLISHA VYNAKTEN".

In addition to this, you can vary the pitch and slope of the voice coming from the Speech Synthesizer to produce a variety from "Donald Duck" to a deep baritone. The documentation for the Speech Synthesizer contains the instructions as does Appendix C of the TE II manual. It follows the form:

// xx yyy where "//" is the preliminary command "xx" is the pitch (from 0 to 63) and "yyy" is the slope (from 0 to 255).

Another thing that you can do with the TE II cartridge is to add inflection symbols and stress points to your speech. The symbols " " and "_" are the inflection symbols while ">" shifts the stress points within the word itself. Since the inflection symbols are non-alphabetic and cause a word break, they must precede the word in the PRINT statement.

Incidently, TI tried to stuff the entire discussion of the TE II speech capabilities into some nine pages in Appendix C of the TE II manual. Considering the space alloted, they did a fair job.

The best way to find out what your Speech Synthesizer will do is to use the short allophone program in Appendix C and experiment with both the pitch and slope and use it to break the words down into their component allophones.

To utilitize the capabilities of text-to-speech in the TE II cartridge, you must first open a file (just like a printer). You will need to use this form:

OPEN #1:"SPEECH",OUTPUT or OPEN #1:"ALLOPHON",INTERNAL

Your decision on which one to use depends on whether you are using text strings (the first) or the allophone file (the second).

There is, however, a rather serious limitation to using the TE II cartridge for your speech programs. While it provides an unlimited speech vocabulary, you are forced to program in the very slow and cumbersome BASIC environment. If you could only use the programming power of XB along with the unlimited vocabulary of the TE II text-to-speech.

Now, we are going to look at the XB cartridge and what speech facility it has in it.

The XB cartridge was designed to access the "373 word" vocabulary that is built into the Speech Synthesizer. It does it with the CALL SAY subprogram. The form that it uses is really quite simple:

CALL SAY("HELLO")

or

CALL SAY(A$) where A$="HELLO"

This is fine if you are using single words but how about the phrases that are in the resident vocabulary. TI really screwed things up on this one --- they never adequately documented how to access phrases like "TEXAS INSTRUMENTS" with CALL SAY. The correct method has been published many times before but here it is "just one more time":

CALL SAY("#TEXAS INSTRUMENTS#")

You can make the phrases that you program more natural by using commas (","), periods ("."), or the plus sign ("+") between the words in the phrase. The comma inserts a slight pause, the period a longer pause and the plus sign links the words together so they sound more natural.

A second subprogram has been supplied with the XB cartridge to make speech access easier. It is the CALL SPGET routine. It provides access to the speech codes used for the individual words or phrases. It follows the form:

CALL SPGET("HELLO",W$)

If you then PRINT W$, you will get the speech codes used to produce the word HELLO. This can be valuable information for creating your own words by concatenating parts of several word strings.

The "373 word" vocabulary that is resident in the Speech Synthesizer is a serious limitation if you have a lot of words that aren't in it. Luckily, just after TI pulled the plug", they released a disk of Text-To-Speech A/L routines that allow unlimited speech capability from the XB cartridge without intruding on any of the programming space. There are three machine language routines on the disk that load into low memory expansion and allow text-to-speech facility. In a program, they would

be entered like this:

```
100 CALL INIT
110 CALL LOAD("DSK1.SETUP","DSK1.XLAT","DSK1.SPEAK")
```

This loads the three routines into low memory expansion. Then, by using CALL LINK, we can access the unlimited text-to-speech.

```
120 CALL LINK("SETUP","DSK1.DATABASE")
```

This line will allow SETUP to load the database of allophones used to create the unlimited vocabulary. This statement only has to be executed once at the beginning of your program. The speech data will remain in memory until the expansion system is turned off. At this point, the text-to-speech system is ready to accept input.

To produce the speech itself, the following format is used:

```
CALL LINK("XLAT","HELLO",A$)
```

"HELLO" represents the text to be spoken (in this case a string constant) but string variables or string expressions can also be used. A$ is the allophone string into which the allophones which comprise the speech data for "HELLO" are placed. A second CALL LINK activates the speech facility.

CALL LINK("SPEAK",A$,xx,yyy) A$ is the allophone return string from the previous CALL LINK statement. The value "xx" is the pitch (from 0 - 63) and "yyy" is the slope (from 0 - 255).

Remember, you can use the stress points and inflection symbols that we talked about arlier to produce a more realistic speech pattern as well.

The only limitation is that you are limited to 128 characters in the CALL LINK statement and the number of allophones returned cannot exceed 255. For longer phrases, simple divide the phrase into tow or more parts.

You can also string allophones together with CHR$(x); where "x" is the allophone number. Remember to use "&" when you combine the allophones into a string.

The character codes between 249 and 255 have special meanings and can also be used in producing speech. Code 249 indicates a secondary stress point, code 250 functions as a sentence break and must be followed by two parameters (the first is the number of stress points before the secondary stress point and the second is the number afterwards), and code 251 indicates that a new default slope value follows. Code 252 indicates a new default pitch value, 253 indicates a rising pitch contour, and 254 is a falling pitch contour. Code 255, which is followed by a byte which gives the pitch parameter, functions as a temporary pitch modification for the next allophone.

I have sent along a demo program that illustrates some of the same basic concepts that we have discussed here in this column. Of course, you can also use speech in A/L and Forth but the A/L is better left to someone with the expertise of Tony McGovern and I understand that there isn't much of an interest in Wycove Forth. Maybe some other time . . .