Vol.7 No.7 September 1989

NH NEW HORIZONS NEWS LETTER

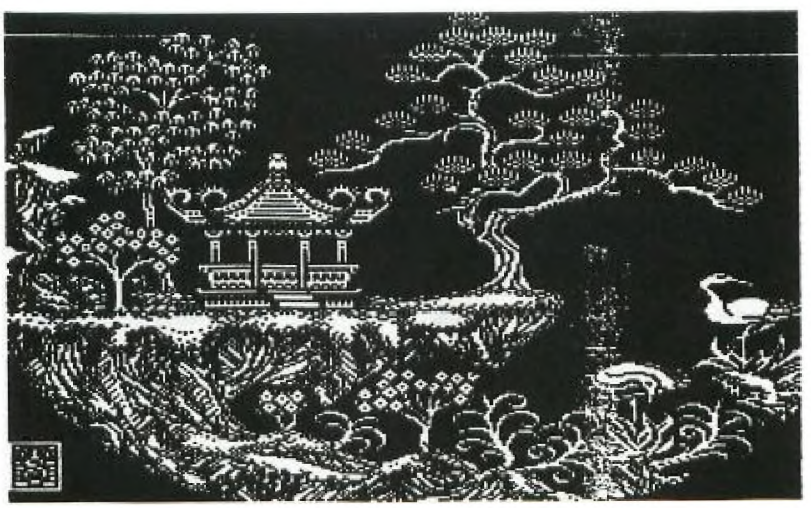
NORTHWEST OHIO COMPUTER CLUB FOR THE TEXAS INSTRUMENTS 99/4A

AND THE MYARC GENEVE 9640 PERSONAL AND HOME COMPUTER

 For more information consult one of the following officers: Yearly Dues $15.00 per Family or Individual.....

THIS MONTHS MEETING SEP.09,1989 SATURDAY AT UNITY CHURCH 12:30 PM.
Behind Wendy's off Secor Road on Executive Dr.

TI-COM BBS.....	1-419-385-7484
PRESIDENT... Mark Maisonneuve ADDRESS 1046 Whll,Maumee Ohio 43567	1-419-893-9314
VICE-PRES... Jo Symington	1-419-474-4128
SECRETARY..Marilyn Schafstall	1-419-882-6870
TREASURER..Earl W.Hoffsis	1-419-475-0461
SOFTWARE LIB.John&Chris Dewey	1-419-475-3871
NEWSLETTER LIB. Burr Mallory	1-419-882-6769
EDITOR....Roger Feinauer	1-517-263-6144
COMPILER..Judy Feinauer	.. ..



ATTN:EARL W. HOFFSIS
N. W. OHIO 99 ERS USERS GROUP
FIRST CHURCH UNITY
3545 EXECUTIVE PARKWAY
TOLEDO, OHIO 43606

STAMP

PRESIDENT'S CORNER

Mark Maisonneuve

Hello again, we are starting the fall season. At this time I regret to announce that I have decided to step down from the Presidentship of New Horizons Computer Club. Due to the demands on my time, I find that I cannot give the attention deserved to the club. Because of this I am now handing over the reins of the presidency to Jo Symington. I do intend to remain a active member of our organization.

This month we will have as always
the club disk, last three issues of the Micropendium magazine, and as schuduled various demos.

At this time I would like to thank all the people who have helped me through the year. So with this closing, bye for now and I hope to see you all at this months meeting.

9,600-bps Modems

My company needs to set up a number of PCs at different locations with direct, point-to-point communications capabilities. To make data transfers faster, we would like to equip the PCs with 9,600-bit-per-second modems.

My question is this: are telephone lines capable of carrying 9,600-bps signals and, if so, what else is involved in setting up a 9,600-bps communications network?

Doug Barcus
Pittsfield, Massachusetts

Yes, you can achieve 9,600 bps over ordinary phone lines. All you'll need is a 9,600-bps modem at each site and software that supports a high-performance file transfer protocol such as Kermit or Xmodem.

It was once true that a 9,600-bps modem could converse only with an identical model from the same manufacturer. No longer. With the advent of CCITT V.32, the modulation characteristics of 9,600-bps modems have been standardized. High-speed modems conforming to CCITT V.32 are now available from Hayes and other modem manufacturers. You can safely mix brands if each one supports V.32 and is full-duplex.

Avoid 9,600-bps modems that don't fully support V.32 or operate in half-duplex mode only. Hayes V-series modems fall into this category. If for one site you buy a modem that uses proprietary technology, you'll have to buy the same type of modem for the other sites.

FRANK & ERNEST

NEW HORIZONS
NEWS EDITOR
ROGER FEINAUER

I'm glad to be back from summer vacation. First off, I have had disaster after disaster. First, in the beginning of this summer my Geneve died and as of this writting is still not working. Tried to get hold of Myarc but get nothing but busy signals on the telephone, but still trying. So, everything this month is done on my 99/4A, GramKracker, and 4 disk drives.

After hooking up my GramKraker I found that the screen was nothing but a multicolor mess. I haven't use it in two years. Also, I couldn't find my manual. Well ,after playing with it for about a week I found a loose wire on the reset switch. There I was up and running again.

Here I sit with over a thousand dallars worth of the Geneve, and Ramdisks in a shoe box waiting for Myarc to answer there phone.

The summer wasn't a total loss. The first thing I did was to set up a disk on drive #4 as I don't use drive number four very much. Set BK to load a program I call PRLD instead of LOAD at drive NO. 4. This program is a menu program that will load about fourty odd programs. You see, the Geneve spoiled me and I miss it. Some of the programs that load from four are TI-Writer,TI-Artist,Masstran ,DSKU,ARC303,SYSTEX,XB EMULATION OF E/A,CALENDER PROGRAM,AND A DIR PROGRAM which will load XB programs and view text prgrams, I also found a catalog program that I rewrote to catalog any disk drives 1-9 and at each page allow you to load either in XB or EA opt5 that prgram. All this and much more on drive four. So you see this summer wasn't a total waste.

The following is the boot program for TI-ARTIST. You will notice that in line 1 that must first load a program called ARTIST, also make sure that you let the system know where your loading it from. This has been tested with a Myarc hard drive as well. Such as WDS1.ART.ARTIST.

In line 2 you will see the statement A$="DSK$." this can easily be anything you wish, but remember the period must end the statment for it to work right. What is happening is after ARTIST is loaded into memory you are inserting what-ever is in A$. To the program ARTIST, this program runs the different modules of TI-ARTIST.

```
1> DISPLAY AT(12,3)ERASE ALL:
"Booting.... Please wait." ::
CALL INIT :: CALL LOAD("DSK4.
ARTIST")
2> A$="DSK4."
3> FOR I=12288 TO 12288+LEN(A
$)/1 :: CALL LOAD(I,ASC(SEG$(
A$,I-12287,1))):: NEXT I :: C
ALL LOAD((12318,0,LEN(A$))::
CALL LINK("START")
```

With the above routine you can load by adjusting lines one and two of the program to any valid store device. I've even made TI-ARTIST run from a hard drive.

Roger

CALL LOAD TO ASSEMBLY AND BACK

by Tom Freeman

This article and the programs that accompany it are another in my intermittent series to help those interested in understanding assembly programs better. You will find XBasic programs that will convert assembly language programs in various formats, which might have one of two purposes. Either you wish to increase the portability, of a program, or you wish to disassemble it to understand what the programmer was doing.

Many programs that use assembly subprograms are published in a "CALL LOAD" format. In other words, the XBasic program directly "pokes" the assembly program into memory, byte by byte. This is done because it might be cumbersome to type in the source code for the assembly program and then assemble it, or you might not have the Editor/Assembler (everyone should, however!). Nevertheless I have published most of my programs this way. The author might also publish the (uncompressed) DIS/FIX 80 object file, but if you have ever looked at one of these, each line is just a long string of numbers and letters that make no sense, and it would be almost impossible to avoid a typing mistake! The CALL LOAD's on the other hand, are full of commas and easily read numbers, so typing them in is easier. However that portion of the program must be run every time the program is run, which takes extra time, so it would be nice to be able to convert them to a real assembly file. Two recent examples of pograms that use this method are: _"Artist to XB" in Smart Programmer, September 1986 - contains two columns packed with CALL LOAD's, and _Improved Unrunnable Basic in Topics, September 1986.

The first program, entitled CL/ASL, that follows this article (I have placed all the programs together, for neatness' sake, so that they could be in 28 column format which looks EXACTLY the way you type it in) provides a method of turning a CALL LOAD XBasic into either a source code file, which can be run through the Assembler to produce an object file, or an object file directly. Thus there are really two programs here - lines 190-280 could be deleted if you only want to make source code, or 290-350 for only object code. I haven't been able to test this program on lots of files, so I suggest you use them both, in case one produces errors. Naturally I have tried to account for all the errors I could think of! One that cropped up was when the CALL LOAD began with an odd address. Assembly files normally insist on even addresses. I compensated for this by backing up to the even address one lower and beginning with the last byte from the previous line. Try this out with a sample two or three line file to see what I mean. The assembler automatically backs up one byte if the AORG or RORG address is odd, and inserts a zero byte first. This would mess up the code, which is why I retained the previous byte.

The only constraints on the input file are that it must 1) be saved in merge format (DIS/VAR 163) not as a program file, 2) contain only CALL LOAD's (delete all other lines and any other statements on the CALL LOAD lines before saving) and 3) only one CALL LOAD(Address,byte,byte,...) per line. The program makes heavy use of a knowledge of how the program lines are tokenized. You can see this for yourself by running the last program in this article on a sample file and

comparing the bytes generated with the list of tokens also provided.

I found one interesting quirk in the way TI handles these assembly DIS/FIX 80 files. Normally the author of a CALL LOAD type program needs to set the REF/DEF table just below 16384 (hex >4000) byte by byte, and then insert the address of the beginning of the table into 8196 (>2004). I originally tried to do this just with AORGs, but the XB loader just won't insert the bytes there even if the assembly file tells it to! CALL LOAD works fine however. I fixed this up by assuming that all code above 16225 is for the REF/DEF table (this leaves room for 20 DEF's and it appears that no one ever has actual assembly code at this location) and then actually construct a real DEF table. Then the loader sets the proper address into >2004 by itself.

Now when the file is ready you can replace ALL the CALL LOAD's by CALL LOAD("DSK1.YOURFILE") where YOURFILE is whatever you named your DIS/FIX 80 file (produced directly by my program, or assembled from the source code it produced). By the way, I lied a little when I wrote above that the assembly program needs to be reloaded every time you RUN the XB program. When a program is finished, the assembly code remains in memory unless you quit or CALL INIT again. So you can add a line to any such program that "PEEK"s at a couple of bytes that you know the value of (do the peeking after the program is run the first time) and then bypass the CALL INIT and the CALL LOAD if the bytes are what they should be. This works with either method of loading the assembly file (CALL LOAD(dis/fix 80 file) or CALL LOAD(address,bytes).

By the way, the program takes quite a bit of time to run, especially if the CALL LOAD's are numerous, but at least it only has to be done once!

The second program, entitled ASL/CL, reverses the process. Why would you want to do this? There are two possible reasons: one might be that you have an XB program and wish to publish it, or list it for a friend. Putting the assembly code into CALL LOAD format makes it all readable in one program. Another reason could be that you wish to have the program on tape for someone who has memory expansion but not a disk drive (my son was originally in this position). The program as listed also is a "double program, as it allows you to construct te CALL LOAD file from a memory range, or directly from a DIS/FIX 80 file. Most object files can be simply loaded from command mode by CALL INIT :: CALL LOAD("DSK1.XXX") and my program then run with the memory range option. [This part of the program runs considerably faster.] WARNING - a few files insert the start address into the ISR hook at >83C4, and will thus auto start. You will need to run the program on the DIS/FIX directly or use a sector editor to change that value (you would find at the end of the file something like 983C4BXXX where XXX is the start address. It should be changed to 000). Please note that the program ends with a statement on the SCREEN that you should type in one or two extra CALL LOAD's. I could have had the program do this, but I didn't get around to it and time is short! [Please note that if the program does use the above auto-start method, then you will need to add one additional CALL LOAD(-3184,x,y) where x and y are the decimal representations of the two bytes following 983C4B above, e.g. if you saw 24F4 than x and y would be 36 and 244].

If you are going to use the memory range option after loading the DIS/FIX file, there is an additional program that will help you, called ORIGINS. Many object files do not load all the bytes in the whole range of address used, but instead leave some blank, to be used later by the program (this is signalled by the BSS directive in the source code). ORIGINS will search for these breaks - actually it just lists al the origins, and you can see if there are large gaps as normally a single DIS/FIX record can only contain about 22 bytes. You can then specify each memory range separately in the program and not wasted a lot of CALL LOAD's unnecessarily.

There is one additional type of assembly file that I haven't mentioned. Some authors have written assembly code, and then "hidden" it in the XB file, using various methods such as Barry Traver's ALSAVE program. You should suspect this when the XB program as listed has more sectors than could be accounted for by the number of lines you see, or if you see a CALL LINK or a CALL LOAD(-31804,x,y) when no assembly file was loaded. The program called HIDDEN will search for the area containing the assembly file and inform you of the range. If you save the ASL/CL program in merge format, and then merge it into the hidden program, you can specify the memory range and produce a CALL LOAD file. I must warn yu however, many of these are quite long and would produce a gigantic CALL LOAD file! You would probably be better off in that case to use SAVE to produce a separate program image file and then DISkASSEMBLE it! (See my article in November 1986 Topics to see how to use SAVE).

Finally the last program is called PRINTMERGE. This will take a MERGE type file and produce a neat listing in compressed format on a printer of each byte of each line and the ASCII representation if possible underneath it. You can do this on a few lines of code to see how program lines are tokenized. If you run it on a single CALL LOAD line for instance you would find the following: the first two bytes represent the line number (multiply the first by 256 and add the second). The rest of the bytes are the tokens, or strings, and the last is always a 0. After the line number bytes you will find: 157 (CALL), 200 (unquoted string), 4 (length of next string), 76, 79, 65, 68 (L, O, A, D) 183 (left parenthesis) 200 (unquoted string) x (the length of the string) x x x (the actual string, in this case the address to be loaded) etc. Have fun with this one, but DON'T use it on large files, unless you have lots of paper!

Ultimately my purpose in writing these programs was to be able to disassemble the CALL LOAD's to understand them. What I did was to produce files that DISkASSEMBLER could read. Reversing the process merely became a challenge! Here's hoping you find these programs useful. Enjoy.

Tom Freeman

"There's your trouble. You're going to laugh when you see all the mistakes you made assembling that unit."

```
**********************************
*                                *
*      SysTex Version 1.0        *
*    Program Hybridization       *
*           Utility!             *
*                                *
*   (C) 1985 By Barry Boone      *
**********************************
```

o If you have not loaded a program (i.e. the DEF table is empty) SysTex will say bad DEF table.

o All three errors will abort Systex

3. Sample session

For this example, have an XBASIC relocatable ml utility ready.

CALL INIT :: CALL LOAD("DSKx.filename") !Loads the program to be saved. RUN "DSKx.SYSTEX"

! SysTex will now ask if you've loaded the assembly to be saved. Type a "Y", then hit ENTER.

! If there are no errors, SysTex will Now prompt you to type in the following:

CALL LINK("SYSTEX")

! At this point, you may LIST to look at the file created.

SAVE DSKx.filename ! Save the file to disk.

! When you RUN this file in the future, it will load the ml, then stop. You may add whatever you wish to the generated program, but you must NOT alter the existing lines (0-9)

* And That's All! *

SysTex is a utility to allow you to save your XB machine language programs/ utilities as XB program files.

Using Systex: ----------

1. Load Machine language file to be saved. The ml should be relocatable. (If absolute origin, see Addendum)

2. Load and run SysTex (Error messages discussed below)

3. Save resulting program to disk.

* And That's IT! *

The resulting program lines (0-9) must NOT be modified in any way! Doing so will destroy the integrity of the program. You may, however, add to the end as much as you like. (See the Example Session at the end)

Addendums

1. Absolute origin files

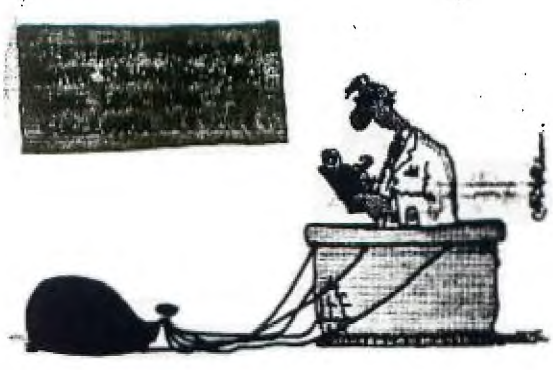
You must find the address of the first byte following the end of the ml program and CALL LOAD it into address 8194. You may then run SysTex.

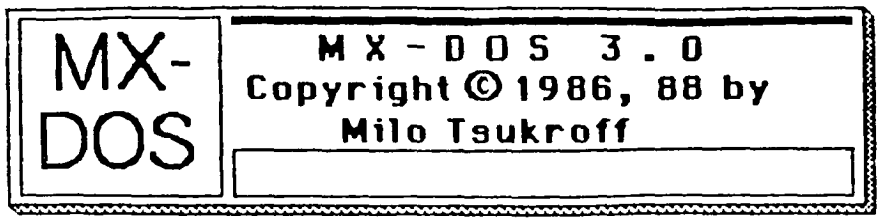
2. Error messages

There are 3 types of errors.

o If you haven't yet done a CALL INIT, SysTex will tell you.

o If you have loaded an AORG file, and haven't done #1 above, SysTex will complain.





MX-DOS 3.0 For TI-99/4A Disk Systems Released

At the January '89 Nutmeg TI-99ers Users Group meeting, Milo Tsukroff released his MX-DOS 3.0 for TI-99/4A disk systems. The MX-DOS 3.0 system is a utility which combines features of a disk manager and an auto-loader.

The MX-DOS 3.0 system allows the average TI-99/4A user to see files on a disk. The user can then run programs, view or print text, and even delete files. MX-DOS 3.0 uses a "MacIntosh"-style graphical interface. The user can use just a joystick to perform nearly all MX-DOS operations. The keyboard is also fully supported.

The MX-DOS 3.0 system is distributed on one single-sided/single-density diskette. Demonstration programs and full documentation are included.

At the Users Group meeting, MX-DOS 3.0 was given out as the 'Disk of the Month'. Milo is distributing it on the 'Fair-Ware' concept, with a fee of $8.00 suggested. This fee includes registration, support, and one free copy of the next major update.

Minimum requirements for MX-DOS 3.0 are a TI-99/4A console; TI Extended BASIC; a single disk drive; and a 32K memory expansion. Additional peripherals supported are joysticks, print[illegible]or monitor, and more than one disk drive.

Additional features for MX-DOS, and speed improve[illegible]ill come when version 3.1 is released. Milo is waiting to se[illegible]ne Fair-Ware registration response is before continuing to impro[illegible]e his product. Even in its current condition, which includes long [illegible]ng times and sluggish response, MX-DOS 3.0 represents an er[illegible]ly easier operating environment to work on disk systems with