THE MISOSYS QUARTERLY

Look at what is in this issue:

- A new EIA-232 driver for LS-DOS with XLR8er,
  by James Cameron
- CL3: Additional Remarks,
  by Roy Soltoff
- PRO-WAM: What's it all about,
  by Roy Soltoff
- HELP to TEXT and Back,
  by Brian Davis
- Fix for Model III SuperScripsit and LDOS directory,
  by MISOSYS
- Update to XLBOOTC or D/FIX,
  by Frank Slinkman

Volume Vi.iv **$10** **Summer/Fall 92**

PRICE LIST effective July 1, 1992

TRS-80 Software

Product Nomenclature	Mod III	Mod 4	Price S&H
AFM: Auto File Manager data base	P-50-310	n/a	$49.95 D
BackRest for hard drives	P-12-244	P-12-244	$34.95
BASIC/S Compiler System	P-20-010	n/a	$29.95 B
BSORT / BSORT4	L-32-200	L-32-210	$14.95
CP/M (MM) Hard Disk Drivers		**H-MM-???**	**$29.95 B**
CON80Z / PRO-CON80Z.	M-30-033	M-31-033	$19.95
diskDISK / LS-diskDISK	L-35-211	L-35-212	$29.95
DISK NOTES from TMQ (per issue)			$10.00
DoubleDuty		M-02-231	$49.95
DSM51 / DSM4	L-35-204	L-35-205	$49.95
DSMBLR / PRO-DUCE	M-30-053	M-31-053	$29.95
EDAS / PRO-CREATE	M-20-082	M-21-082	$44.95 D
EnhComp / PRO-EnhComp Diskette	M-20-072	M-21-072	$23.98
Filters: Combined I & II	L-32-053	n/a	$19.95 B
GO:Maintenance	n/a	M-33-100	$39.95 B
GO:System Enhancement	n/a	M-33-200	$39.95 B
GO:Utility	n/a	M-33-300	$39.95 B
HDPACK: Disk De-fragger	**n/a**	**M-33-400**	**$39.95**
Hardware Interface Kit	n/a	M-12-110	$24.95
HartFORTH/PRO-HartFORTH	M-20-071	M-21-071	$39.95 B
LDOS/LSDOS Reference Manual	M-40-060	M-40-060	$30.00 B
LDOS/LSDOS BASIC Reference Manual	M-40-061	M-40-061	$25.00 A
LDOS 5.3.1 Diskette - M1	M-10-110	n/a	$15.00
LDOS 5.3.1 Mod1 Upgrade kit	M-10-133	n/a	$39.95 B
LDOS 5.3.1 Diskette - M3	M-10-130		$15.00
LDOS 5.3.1 Mod3 Upgrade Kit	M-10-333	same	$39.95 B
LED / LS-LED	L-30-020	L-30-021	$19.95
LB Data Manager-M4 (Ver 2.2)	n/a	M-50-510	$99.00 D
LS-DOS 6.3.1 Upgrade Kit - M4	n/a	M-11-043	$39.95 B
LS-DOS 6.3.1 Diskette - M4	n/a	M-11-243	$15.00
LS-DOS 6.3.1 Upgrade kit - M2/12/16		M-11-002	$39.95 B
LS-Host/Term	n/a	L-35-281	$39.95
LS-UTILITY	n/a	L-32-150	$24.95
MC / PRO-MC	M-20-064	M-21-064	$79.95 D
Mister ED	n/a	M-51-028	$39.95 B
MRAS / PRO-MRAS	M-20-083	M-21-083	$59.95 D
PowerDot (Epson or Tandy)	P-32-21?	n/a	$19.95
PowerDraw	P-32-220	n/a	$19.95
PowerDriver Plus (Epson).	P-50-200	P-50-200	$17.95
PowerMail Plus	P-50-003	P-50-004	$39.95 D
PowerMail Plus TextMerge	P-50-100	P-50-100	$15.00
PowerScript	P-50-142	P-50-142	$24.95 B
PRO-WAM	n/a	M-51-025	$74.95 D
PRO-WAM Toolkit	n/a	M-51-225	$29.95 B
Programmer's Guide DOS 6.	n/a	M-60-060	$20.00 B
QuizMaster	L-51-500	n/a	$19.95
RATFOR-M4		M-21-073	$59.95 D
RSHARD - R/S HD driver	M-12-013	same	$29.95
ST80-III, ST80-PBB, ST80-X10	P-35-300	n/a	$39.95 B
SuperUtilityPlus	P-32-132	P-32-104	$44.95 D
SuperUtilityPlus CMD file diskette	P-32-832	P-32-804	$20.00
Supreme HD Driver (PowerSoft-RS)	P-12-113	P-12-113	$34.95
TBA / LS-TBA	L-21-010	L-21-011	$19.95 D
THE SOURCE 3-Volume Set	n/a	L-60-020	$40.00 D
Toolbox/Toolbelt	P-32-203	P-32-245	$24.95 B
UNREL-T80	same	M-30-054	$29.95
UTILITY-I	L-32-070	n/a	$19.95
XLR8er Softawre Interface Kit (M3 mode)		M-12-X10	$20.00 B

TRS-80 Game Programs

Bouncezoids (M3)	M-55-GCB	$14.95
Crazy Painter (M3)	M-55-GCP	$14.95
Frogger (M3)	M-55-GCF	$14.95
Kim Watt's Hits (M3)	P-55-GKW	$9.95
Lair of the Dragon (M3/M4)	M-55-021	$19.95
Lance Miklus' Hits (M3)	P-55-GLM	$19.95
Leo Cristopherson's (M3)	P-55-GLC	$14.95
Scarfman (M3)	M-55-GCS	$14.95
Space Castle (M3)	M-55-GCC	$14.95
The Gobbling Box (M3/M4)	M-55-020	$19.95

MSDOS Game Programs

Lair of the Dragon	M-86-021	$19.95

Hardware

Power Supply, 40WT Astec AC8151	H-PS-A40	$40.00	D
Power Supply, 68WT Astec AA12310	H-PS-A68	$50.00	D
Floppy Disk Controller M3/M4	H-MM-FDC	$45.00	F
Double Density Controller (DDC) M1	H-MM-DDC	$45.00	F
RS232 Serial Card M3/M4	H-MM-SPC	$45.00	F
RS232 Serial Card Kit M3/M4	H-MM-SPK	$50.00	F
TeleTrends TT512P modem (M4P)	H-4P-512	$74.95	E
Floppy drives (5.25" 360K 1/2 ht)	H-FD-360	$75.00	D
Floppy drives (3.5" 720K 1/2 ht)	H-FD-720	$85.00	B
Floppy Drive Case (2-1/2 ht drives)	H-FD-2SV	$60.00	F
MSCSI HD kit e/w clock, 20Meg M3/M4	H-HD-020	$450.00	?
MSCSI HD kit e/w clock, 40Meg M3/M4	H-HD-040	$575.00	?
Aerocomp HD - 20 Meg M3/M4	H-MM-020	$400.00	?
Aerocomp HD - 40 Meg M3/M4	H-MM-040	$500.00	?
MSCSI Hard Drive joystick port option	H-HD-JSO	$20.00	
Hard drive: Seagate ST225 (20M)	R-HD-020	$200.00	F
Hard drive: Seagate ST251-1 (40M)	R-HD-040	$320.00	F
Hard drive: Seagate ST-157N (SCSI)	R-HD-S40	$300.00	F
Cable: dual floppy extender	H-FD-2EX	$18.00	
Cable: 4Ft floppy (1 34EDC each end)	H-FD-C04	$12.50	
Cable: 4Ft M3/M4 printer	H-RC-PM4	$20.00	
Cable: 4Ft Radio Shack hard drive	H-HD-CT4	$20.00	
Cable: 4Ft MISOSYS hard drive	H-HD-C04	$22.50	
Cable: 26-1069 internal floppy	H-FD-2NG	$20.00	
Cable: 26-1069A/26-1080 internal floppy	H-FD-2GA	$20.00	
Cable: 26-1080/A internal floppy	H-FD-24P	$20.00	
Cable: drive power Y	H-HD-CPY	$5.00	
Cable: XT hard drive set	H-HD-CXT	$5.00	
Cable: Custom IDC ribbon (M3/M4/M2)	?-??-???	varies	
Standby Power System: 200VA	R-PS-200	$199.00	?
Standby Power System: 450VA	R-PS-450	$399.00	?
HD Controller: Adaptec 4010A	H-HD-CA4	$75.00	D
HD Controller: Xebec S1421A	H-HD-CX2	$75.00	D
HD Controller: WD1002S-SHD	H-HD-CW2	$75.00	D
T80 to SCSI host adaptor	H-HD-MHA	$75.00	D
ZOFAX 96/24 **Fax/Modem** (PC XT/AT)	R-Z1-FAX	$125.00	G
Infochip Systems Expanz! (PC)	R-IC-EXP	$150.00	G
DJ10 Tape Backup (PC)	R-TD-D10	$199.00	G
DJ20 Tape Backup (PC)	R-TD-D20	$265.00	G
AB11 Tape Adaptor (PC)	R-TD-A11	$45.00	D
KE10 External tape adaptor/case (PC)	R-TD-K10	$110.00	F
Tadiran TL-5296 AT 6V lithium battery	R-PB-TL6	$19.95	B

MSDOS Software

LB Data Manager 2.2	M-86-510	$99.00 D
DED-86 [Disk/Memory sector editor]	M-86-020	$29.95 D
RATFOR-86	M-86-073	$59.95 D
HartFORTH-86	M-86-071	$59.95 D
SAID-86 [Text Editor]	M-86-040	$29.95
Super Utility PC	P-86-407	$29.95 B
TRSCROSS (transfer <> Mod III/4	P-86-212	$89.95 B
FM-86 (File Manager)	L-86-050	$29.95
Lair of the Dragon	M-86-021	$19.95

The Fine Print

Freight codes: A = $3.50; B = $4.00; C = $4.50; D = $5.00; E = $5.50; F = $6.00; G = $7.00; H = $12.00; ? = varies; All unmarked are $3.00 each; Canada/Mexico add $1 per order; Foreign use US rates times 3 for air shipment. Virginia residents add 4.5% sales tax. We accept MasterCard and VISA; Checks must be drawn on a US bank. COD's are cash, money order, or certified check; add $4 for COD.

MISOSYS, Inc.

P.O. Box 239, Sterling, VA 20167-0239

703-450-4181; Orders only: 800-MISOSYS

The MISOSYS Quarterly is a publication of MISOSYS, Inc., PO Box 239, Sterling, VA 22170-0239, 703-450-4181.

THE MISOSYS QUARTERLY

subscription rate information

Each issue of TMQ has information on MISOSYS products, programs and utilities, patches, significant messages from our CompuServe forum, and articles on programming. Not only that, TMQ will keep you up to date with information, news, and announcements concerning our entire product line and related machine environments. Subscription cost varies by rate zone as follows:

A = $25; United States via 3rd class bulk mail
B = $30; Canada, Mexico, via 1st Class
C = $32; Colombia, Venezuela, Central America via AO Air
D = $35; South America, Europe, & North Africa via AO Air
E = $40; Asia, Australia, Africa, Middle East via AO Air

TMQ Toolbox

The MISOSYS Quarterly is published using the following facilities:

The hardware used to produce the "camera ready" copy consists of an AST Premium/386 computer (20 MHz) with 9 Megabytes of RAM, a Seagate ST4096 80M HD, ST251 40M, Expanz! card; a CMS DJ10 tape backup, a NEC Multisync II monitor driven by a Video Seven VGA card, an AST TurboScan scanner (Microtek MS300), and a NEC LC-890 PostScript laser printer.

Text is developed, edited, spell-checked, and draft formatted using Microsoft WINWORD Version 1.1; Submissions on paper and letters are scanned and converted to text using ReadRight optical character recognition software by OCR Systems. Final page composition is developed using PageMaker 4.0 by Aldus.

Table of Contents

The Blurb

TMQ to Continue	2
Points to Ponder	2
Trade-in Policy	3
In this issue...	3
TMQ Schedule	3
MISOSYS Forum	3
DISK NOTES 6.4	3
EnhComp BASIC Compiler	4
MS-DOS Products	4
Power Supplies	5
HDPACK for HD's	5
Address Change	5
FAX Number	5

Letters to MISOSYS

Pro-EnhComp and USING	6
2 sided drives on 4P	6
Aerocomp Driver disk	7
Splitting a File	8
Update to XLBOOTC\D/FIX	8
Model III SuperScripsit	8

Inside TMQ

A new EIA-232 driver for LS-DOS	10
PRO-WAM	30
PRO-NTO: TRSDOS 6.X's Sidekick	36
HELP to TEXT & Back	43

List of Advertisors

MISOSYS, Inc.	IFC,51,52,IRC,RC
Pacific Computer Exchange	50
TRSTimes magazine	51

List of Patches in this Issue

XLBOOTE/FIX (Model 4)	8
SCR17/FIX	9

TMQ to Continue

If you haven't been stuck in a *worm hole* lately, you have noticed that *The MISOSYS Quarterly* took on a new look by shifting from the glossy "kromekoat" cover to a black separately affixed perfect binding, and migrated to a colored cover stock printed in black. The change reduced production costs thereby keeping TMQ production in line with the subscription levels, and provided the means to continue publishing. I received one letter requesting that I revert back to plastic film wrap or a heavier cover stock as the Postal Service did their best to crunch up his issue. There's two reasons why I am avoiding the plastic film for domestic mailing: one is cost and time to wrap; the other is that plastic wrap rarely benefits the environment as it gets deposited in the landfills. If your issue is grossly mutilated, let me know and I'll attend to that.

And now for the same price of a subscription as it has been the past six years, look forward to TMQ Volume VII. You can save me the cost of a renewal letter by getting your renewal in early! Check your mailing label; if it has a "**92/08**" after your name, then this issue is the last one of your subscription.

Points to Ponder

Talk about dense, Supercomputer Systems, Inc., and startup funded by IBM and headed up by Steve Chen, a Cray Research former computer designer, recently released some details of its first design - the SS-1 supercomputer. SSI has formulated circuit boards of 78 layers using line widths of 63 microns. Via holes are drilled with CO_2 lasers.

The Blurb

by Roy Soltoff

Recollecting my Ham Radio days, I seem to remember the large size of the power tubes used in the RF amplifier - and that was for a tube generating about 250 watts of RF output. Certainly times have changed, but transistors were never known to be power demons. However, Phillips recently released a silicon bipolar transistor capable of 750 watts of output power at up to 1,150 MHz when operating in a class-C configuration using pulsed transmission at a 10% duty cycle. In short-pulse (1% duty cycle) applications, it generates up to 650 watts up to 1 GHz.

Not only did General Motors have a very bad year, but DEC, the stalwart opposition to IBM in the mini-computer arena, lost $2.8 billion in its last fiscal year. Since 1989, DEC cut its personnel by 23,000. Possibly in light of this poor performance, DEC's founder and president, Kenneth Olsen, retired. DEC's CEO also stepped down. I don't know about you, but I believe the days of the profitable mainframe computer companies are numbered. Judging from the shake-ups going on in the micro world, the viability of some of those hardware companies is also questionable. Even the X86 CPU competition is starting to crumble. C&T is now exiting the 486 and next-generation clone business; they're also dropping their 386SX line, keeping only their Super386DX and SuperMath 387.

Remember how computers were supposed to create the *paperless* office? Well laser printers and faxes took a dim view of that! When I was growing up, the inventor of the pop top soda can was considered a genius, having done away with the *church key*, until those tops started littering the highways, byways, and oceans. That problem was solved with the pop top which remained on the can. Now laser printer cartridges are being stuffed into landfills in record numbers. But there's still room for innovation. Kyocera has now released a new type of LED scanner array printer (just another form of what folks commonly call a laser printer). But this one uses a fine-grain ceramic toner which polishes the print engine resulting in a much longer drum life; Kyocera is offering a 300,000-page warranty on the print drum. Considering your typical Canon cartridge is good for about 4000 copies, that's a 75-fold increase in drum life. The $2400 list FS-1500A printer supports Postscript, PCL5, and HP Laserjet II. Although it won't do anything to reduce the paper-laden office, at least the drums won't fill up our landfills as fast as current printers.

Here's more innovation now from AT&T, my former employer. They have a new low-cost voice recognition module destined to provide voice dialing of mobile cellular telephones, as well as applications in answering machines, modems, and consumer appliances. Toaster, do your stuff!

Finally, on the copyright scene, there's been some interesting rulings.

First, Lotus won out over Borland in the 123-suit against Quatro Pro. Judge Keeton ruled that Borland illegally incorporated parts of Lotus' 1-2-3 spreadsheet in that old *look-and-feel* case. Monetary damages have yet to be ascertained as I write this; however, as expected, Borland is appealing.

Second, on the opposite side of the fence, the 2nd Circuit Court of Appeals upheld a decision involving Altai against Computer Associates which severely restricts copyright protection for the structural el-

ements of a computer program. The decision noted that a program may be composed of numerous submodules that may express many different ideas, and consist of both text and behavior. The judge ruled that the "behavior" is not a proper subject for copyright protection; common ideas, structures, and standard techniques should be filtered out before deciding what portion of a program, if any, deserves to entertain a copyrightable expression.

And thirdly, in April, a U.S. District Court Judge Barbara Caulfield ruled in a reverse-engineering case, "if the process of reverse-engineering entails the duplication of the copyrighted work and the recasting or transformation of the object code into a form more intelligible to humans, it may infringe on the copyright owner's exclusive rights." The case involved Accolade disassembling the object code of a Sega game cartridge to understand how to make cartridges which would work on Sega's game machine. Sounds like the same prohibition on disassembling, for instance, a DOS to understand how to create a competing DOS which operates the same way.

Trade-in Policy

The policy for trade-ins of an equivalent non-MISOSYS software product for a MISOSYS-published software product is to just send in an original *Table of Contents* page with the trade-in fee which is 50% of the price of our product. So for LB 2.2, trade in any other database product and you can purchase LB or LB-86 for $49.50 plus S&H. How's that for a deal? It doesn't matter for what system or operating environment your trade-in was designed for. This offer does not extend to products re-sold by MISOSYS.

In this issue...

There are some exciting articles in this issue. From James Cameron is a serial driver for the XLR8er ASCI ports. His use for the driver turned the Model 4P into a four-user machine with a special purpose in mind. From Brian Davis is a series of programs in BASIC which can be used to generate DOS HELP files, as well as turn exciting HLP files into ASCII for your own editing. With the interest in windowing going on these days, I have been asked to expand on the capabilities of or PRO-WAM window and application manager. An in-depth look at that product is here. These were all big articles, so big that the issue grew past 48 pages. You readers got a bonus.

Next Volume starts our focus on the C language. Look for the first issue to begin with a revised update of an old six-part series originally written by Earl "C" Terwilliger for the old *LSI Journal*. With that coverage will be a series of programs by Rich Deglin which introduce an environment facility to LS-DOS.

Finally, if someone is interested in beginning a series of articles on FORTH - for either MS-DOS or TRS-80, MISOSYS will provide you with a free copy of either HartFORTH for the Model I/III or Model 4, or even HartFORTH-86. And for those who want to play around with FORTH but are not interested in writing about it, I'll be offering our FORTH programs at one-fourth the regular price. That's $9.98 + S&H for the Model I/III or Model 4 version, and $14.99 + S&H for the MS-DOS version, HartFORTH-86.

TMQ Schedule

The MISOSYS Quarterly is mailed approximately every three months. This issue should be mailed less then three months since the issue VI.iii was mailed, so I'm hanging in there - time-wise.

Note that your mailing label usually has the expiration date of your subscription. For instance, those with "**92/08**" complete their subscription with this issue. The renewal fee to continue with the next volume is covered on page 1.

MISOSYS Forum

I sponsor a forum on CompuServe. You can reach some "experts" on TRS-80 and MS-DOS subjects by dialing in, then GO PCS49, or GO LDOS.

The forum contains many programs to download, as well as lively discussions which thread through the message system. You can direct a message to me at 70140,310. Post a message in private if you don't want it "broadcast"; some folks even send me orders via a PRIVATE message.

DISK NOTES 6.4

Each issue of *The MISOSYS Quarterly* contains program listings, patch listings, and other references to files we have placed onto a disk. Where feasible, the text accompanying an article is also on DISK NOTES. DISK NOTES 6.4 corre-

sponds to this issue of TMQ. The disk is formatted for TRS-80 LDOS/LS-DOS users at 40D1 (that's 0 tracks, double density, one sided). If you want to obtain the patches and the listings, you may conveniently purchase a copy of DISK NOTES priced at $10 **Plus S&H. The S&H charges are $2 for US, Canada, and Mexico, $3 elsewhere.**

EnhComp BASIC Compiler

I expect to be doing some improvement to my BASIC compiler, EnhComp. As a matter of fact, I recently completed the first phase. Here's a little background. EnhComp was originally written by Phil Oliver, the author of Scarfman, EnhBAS (an enhanced BASIC add-on to interpreter BASIC), and many other Model I/III products back in the early days of the TRS-80. Phil even wrote his own assembler, ZED, but it was never released for sale. MISOSYS acquired publishing rights to Phil's BASIC compiler a number of years ago and continued to improve it (i.e. rid it of bugs, and adapt it to the Model 4 environment).

One of the big problems I had with EnhComp was that (1) the assembler used to assemble all the modules was written only for the Model III mode; and, (2) it did not support conditional assembly. Thus, the Model III EnhComp and the Model 4 EnhComp had to be assembled from two completely different sets of files. That included all of the programs supplied with the product: the S/CMD supervisor, CED/CMD editor, BC/CMD compiler, REF/CMD cross-reference utility, and SUPPORT/DAT relocatable support module library. That represented a lot of files. Since the assembler did not support conditional code blocks, there was a set of files for each operating environment. That caused lots of problems.

Nothing has been done to EnhComp for a few years because of the pain it took to generate a new version for each environment. I always had a long-term plan to revise the assembler, and groom the files to a ingle set before I started on the next release. I didn't even have the source code to the assembler which I had to get from Phil. Since the last issue of TMQ was mailed, I began work on this project.

Just recently, I completed the first phase which was (1) reworked the ZED assembler to operate in Model 4 mode, (2) added code to support IF/ENDIF conditional assembly, (3) reworked all EnhComp modules to utilize conditional code blocks to support the Model I/III or Model 4 version generation, and (4) compared all programs generated (for both Model III and 4) to ensure that the program files were identical to what is currently being shipped. That was some job. Now I have one big set of files used to generate both versions.

Phase 2, already begun, is to investigate a few bugs, fix them, then work to add more Model 4 interpreter BASIC compatibility. For instance, back in TMQ issue IV.ii, Michael Dauphin reported a problem with PRINT USING incorrectly formatting and/or rounding a floating point number less than 0.1. I spent a few days figuring out the problem and working up a solution (the code is sparsely commented and relatively to understand). But that problem is now fixed. In October of last year, Rick Jones discovered another snag in USING when printing a formatted string where the string length was less than the string field specified by "%spaces%". I just fixed that up. The last report still *hanging*, was a difficulty Ralf Folkerts was having with XFIELDed files. I'm looking into that right now.

Ralf, in fact had a few recent suggestions: "I think the CED could be a bit redesigned to not default to line numbers but editor numbers (or give it a 'configuration' facility). The EnhComp's printing could be improved (the print directive and the REF utility). It's some time ago that I last used them, but I think they defaulted to 66 Line Paper or so. It would be great if they were reading the FORMS parameters (if FORMS is active) and then use this data for printed line/lines per page (the Tandy COBOL compiler does this ... works great)!"

I have no plans at this point, to introduce features beyond that available to BASIC4. Since EnhComp is more or less extensible with its in-line assembler and user-defined commands, it is certainly possible for any programming user to increase the support (sort of like what I did with the SETEOF and SOUND commands. But I am open to suggestions.

Incidentally, anyone who needs the few fixes noted above, can send me a disk and I'll put the revised SUPPORT/DAT libraries on them.

Don't forget that with our newly published "LDOS™ & LS-DOS™ BASIC Reference Manual", which covers the interpreter BASIC which is bundled with LDOS 5.3.1 (even the ROM BASIC portion), the interpreter BASIC which is bundled with LS-DOS 6.3.1, and both Model I/III-mode and Model 4-mode EnhComp compiler BASIC, you can purchase the disk version of EnhComp for $23.98 plus $1.50 S&H when purchased along with a BASIC Reference Manual, or the disk version by itself for $29.98 plus $3 S&H if purchased separately. If ordering the EnhComp disk, please note which version: Model I/III or Model 4!

MS-DOS Products

MISOSYS is a reseller of products purchased from Ingram Micro; thus, we have access to a huge array of MS-DOS products. So if you are looking for some hardware or software to go with your MS-DOS system, why not get in touch with us for a quote. Call, write, or FAX.

Power Supplies

MISOSYS now stocks replacement power supplies for Model III or 4 computers. The Astec AC8151-01 40-watt supply provides +5V @ 2.5A, +12V@2.0A, and -12V@0.1A. It's size is 6.25"x4"x1.75"; mounting holes are 3.125"x4.75". The Astec AC12310 68-watt supply provides +5V @ 7.3A, +12V@2.5A, and -12V@0.1A. It's size is 7.69"x4.125"x2"; mounting holes are 3.75"x7.25". This supply is a direct replacement for the Tandy Model 4 power supply (Tandy's was based on the Astec design). The 40 watt supply is $40 and the 68 watt supply is $50. S&H for either is $5

HDPACK for HD's

MISOSYS announces the release of a new utility for hard drive users. HDPACK, priced at $39.95 + $3 S&H, will re-pack all files on a hard drive so that they take up a minimum number of directory extent fields. If you invoke a DIR command, you will see the number of extent fields in the "Ext" column. Anytime this field is greater than one, there is the probability of increasing the access time of a file, were the number equal to one. A single extent field can hold a maximum of 32 granules. Multiply the typical 4K or 8K granule size, and you see that files up to 256K could be stored using one extent field. And when your file takes up more than four extent fields, it actually uses another directory entry (i.e. wastes an entry usable for another file).

Random access files using more than four extent fields require additional access time as the in-memory file control block holds data on only four extents at a time. For years, MISOSYS has been asked for a utility program such as HDPACK. Now its here. Fail safe, and guaranteed to work. It can even be used on a floppy, if you really want to do that. HDPACK presents a *graphical* image of the drive's file mapping using discrete characters for the directory, free granules, in-use granules, and fixed-position granules. Once a granule has been re-positioned on the drive, its corresponding visual map position is checked off providing a visual indication of its progress. It takes only minutes to repack a 10M partition.

Because it is important that your directory contain correct information to begin with, HDPACK includes a copy of DIRCHECK - our directory checking and correcting program.

Note that because of the memory requirements for running HDPACK, it is available only in the Model 4 operating mode. However, since the directory structure of Model 4 LS-DOS and Model I or III LDOS is identical, a drive running under LDOS can be packed by connecting the drive to a Model 4 and bringing it up in the Model 4 mode under LS-DOS.

Address Change

The United States Postal Service has seen fit to change the ZIP code of our Post Office Box (mailing address) effective July 1, 1992. The new ZIP is 20167. Please update your records; I don't want to lose any of your orders.

FAX Number

If you want to reach us by fax, try 703-450-4213.

ST80-III Version 2.50

I have received a few calls lately which reflect confusion on what is included in the ST80-III package listed in the TRS-80 product catalog. The $39.95 product price includes ST80-III version 2.50, the most recent and most powerful version ever generated by Lance Micklus, as well as his Personal Bulletin Board (ST80-PBB) and Host Interface (ST80-X10).

PRO-WAM Special

This issue of *The MISOSYS Quarterly* provides in-depth coverage of the PRO-WAM Window and Application Manager. To coincide with this coverage, the following special pricing is in effect **for TMQ subscribers only**:

- **PRO-WAM** at $37.48

or

- **PRO-WAM and MisterED** at $74.95

or

- **MisterED** at $19.98

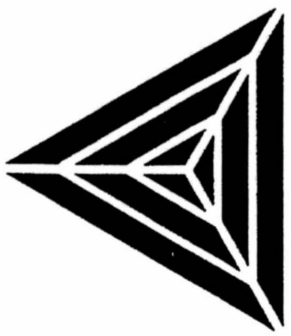
or

- **Programmers Toolkit** at $14.98

The user-contributed wammies noted in the PRO-WAM article are all available in run-time form with on a single diskette for $10 + $2S&H ($3 outside of North America). The source code to each file is available on another diskette. The application source diskette is also available for $10 + $2S&H ($3 outside of North America).

- Order any one of the above PRO-WAM specials, and receive your choice of either runtime or source diskette of wammies free.

Letters to MISOSYS



Pro-EnhComp and USING

Fm Rick D. Jones: Roy, Please help me with this little problem I have with Pro-EnhComp.

The following excerpt from a program I'm writing has wrong output when using the USING function as you can see; the columns do not line up. Is there a fix or procedure I'm missing? I have "Ver. 2.6a - 01-Mar-88"

REC$(I,1) is loaded with variable length strings. REC$(I,2) is loaded with a one character binary number 0-255. program excerpt:

```
REC = REC - 1
RECLEN = 0:CLOSE
A$="'%                          %'
###"
B$="Record Length:
###"
FOR I = 1 TO REC
PRINT USING A$; REC$(I,1),
ASC(REC$(I,2))
RECLEN = RECLEN +
ASC(REC$(I,2))
NEXT I
PRINT "
======"
PRINT USING B$; RECLEN
END
```

The single quotes are just for the visual reference of this problem. I couldn't find any references in the manual, readme/txt or the last few TMQ's on the stripping of trailing spaces with the USING function

Fm MISOSYS, Inc: Rick, I verified what you discovered with a similar program. Apparently you are the first to discover that bug in EnhComp, which is why it was never fixed before. The problem you happened upon occurs when the string expression is printed via a USING with the output controlled by the format "%spaces%", and the string expression length is smaller than the space field. Interpretive BASIC right-fills the field with spaces - as it should. EnhComp neglected to add the required number of spaces.

In order to fix up the problem, I had to add some code to the USING support routine and re-assemble SUPPORT/DAT library. I'm in the process of updating EnhComp (see The Blurb), so if you can wait for the next release, please do. If you need the fix right away, let me know and I can supply you with the interim revised SUPPORT/DAT library file.

2 sided drives on 4P

Fm Robert Hengstebeck: I need a bit of help again. This time I am upgrading my dad's computer, a Model 4P from dos 6.3 to 6.3.1; and for some reason, I cannot get the format command to recognize that the 4P has double sided drives. I have a 10 meg hd and my dad has a 15 meg hd. I am trying to do the update, without changing the configuration of the 15 meg hd. Originally his system had only single sided drives, and his system probably has some parameters buried in it, to only reflect that. But because the hd's are configured differently, I must give preference to his system. So how do I change his system to see the double sided floppy drives, especially with using the 'SYSGEN' command to save a new configuration? It seems to me that the answer lies in the use of the 'SYSTEM' command, but my books show only the options to change the floppy drive access time, eg 6 ms.

I put in the double sided drives in the 4P, and also made up a new cable for the drives. There seems to be no problems with the drives in reading and writing to double sided diskettes in the machine.

What is strange to me, is that I can boot up the 4P, with the diskette that I use to boot

up my computer, and then when the format command is used, the computer will ask me, if I want to format (1) or (2) sides. But when I boot up on the modified boot disk for the 4P, it will not give me that option when formatting.

The command 'FORMAT :x (sides=2)' does work. But I would rather the format command ask my dad, how many tracks, what density, and how many sides does he want formatted. So how do I get the system to recognize that the 4P has the option of two sided drives and to act accordingly? What I can't figure is why the backup of my bootup diskette to the 15 meg hd, and then the backup of the 15 meg hd to the 4P bootup diskette, ends up with the format command operating differently. So what is my next step?

Fm MISOSYS, Inc: Robert, The answer is in the setting of the "L" flag (that's the letter L). According to my information, the number of sides prompt in FORMAT will be inhibited unless bit 5 of the LFLAG is set. Looking at *THE SOURCE* on page 184 of *The Utilities*, I see line 6970 testing BIT 5 of the L-flag. Bit 5 set is a value of 20H. The DOS will not ask the # of sides query in FORMAT if bit 5 is reset. You can determine this setting by issuing a command:

MEMORY (A="L").

Check the resulting value. I'll assume that the value of bit 5 is a zero. Next, calculate the new value of the byte adding 32. Then issue the command: MEMORY (A="L",B=newval). You should then SYSGEN to the BOOT drive after correcting the value of the suspected bad-value Lflag.

Actually, the value should normally be X'21' to prompt for sides and step rate, but inhibit the FLOPPY/DCT 8" query.PC is a nightmare

The following letter which appeared in a major electronics industry newspaper (Electrical Engineering Times, June 22, 1992) is reprinted here with the permission of its author.

Fm Mark D. Pickerill, Monterey, CA: I read with considerable amusement Mr. Andrew Gray's letter in the June 8 issue (see page 32). I could not agree more, but Mr. Gray neglected to mention that not only are we bogged down with the inferior X86 architecture, but we are also bogged down with the silliest implementation imaginable of said architecture: the PC.

Anyone who has taken an unbiased, objective look at the PC will agree: It is nothing more than a glorified TRS -80 (Model I, not the nicely integrated Model III), with the expansion-slot idea of the Apple II + thrown in.

From an architectural point of view, it was designed with the same Basic in ROM, cassette port, CPU power-robbing memory-mapped video, and useless interrupt-driven real-time clock that had to be set each time power was applied. All subsequent "improvements" have been kludges of attempts to get around the limitations of the architecture, the most glaring of which is the gap in user RAM to accommodate memory-mapped video and ROM.

From a mechanical standpoint, the PC is also a kludge. When the PC was introduced, the microcomputer industry was moving toward fully integrated systems, such as the above-mentioned TRS-80 Model III, the Northstar Advantage and the Superbrain. The PC was a throwback to the 1977 introduction of the TRS-80: three ugly boxes connected with cables.

From a software standpoint, the PC is an absolute nightmare. The first converts were TRS-80 and Apple programmers, who were used to employing poor programming practices, such as direct hardware control vs. OS calls. The CP/M community, which laughed at the PC for the kludge it was, did not become involved with it until these poor programming practices were firmly entrenched.

Someone once told me that CP/M stood for "Can't Prove it by Me." Although I have been involved with CP/M for many years, I agree totally . But since MS-DOS is nothing more than enhanced CP/M, I feel MS- DOS really stands for "More of the Same Dumb Old Stuff". MS-DOS is a bastardized cross between CP/M and TRS-DOS, with the worst features of both and the best features of neither.

Where we do we go from here? With the crippled X86 architecture, the ridiculous PC architecture and antique OS, it is hard. Windows? Forget it. This is a kludgey attempt to place the Mac environment on top of another kludge of an OS that in turn is a clone of the kludgey CP/M, which in turn is a clone of the old TOPS 10 operating system of the 1960s. As a result, performance is dismal. It takes a lot of horsepower to run this at any decent speed, because of the many layers of kludgey software.

The Mac, designed for a similar environment, does its job well, as it was designed that way from the start. OS/2? Maybe. While it is definitely a better implementation and has modern features one would expect, it still has to deal with the dismal PC architecture, as well as retain some software compatibility. So where we go from here?

Aerocomp Driver disk

Fm MISOSYS, Inc.: Ron Miller had a problem in trying to access our Aerocomp driver disk using LDOS 5.1.4 (!). Seems that most of the files had password protection. Here's the snag; the problem is minor. If you have one of these disks prepared by MISOSYS, the following provides corrective action.

The files should work properly over on LDOS 5.1 once you take care of what 5.1 thinks are passwords. The disk containing the Aerocomp drivers is an x.3 disk. Ap-

parently the files had owner passwords; however, the access level is FULL access. Thus, an x.3 DOS - either LDOS 5.3 or LS-DOS 6.3 will consider the files as having no passwords. Under LDOS 5.1, you will have to remove these extraneous passwords with the ATTRIB command. Short of upgrading, you can get rid of the *assumed passwords* by using a command under LDOS 5.1 such as **ATTRIB filespec.P3UF:d (o="",u="")**. This will change the password of each file specified to blanks. Obviously, you only have to take care of the files you need to use.

Splitting a File

Fm MISOSYS, Inc: I had a recent customer who had purchased the TRSCROSS and LB86 products to migrate a Profile database to MS-DOS running with LB86. The problem was that his Profile database was stored on a hard drive and the /DA1 data file exceeded the size of his floppy disk. Thus, he couldn't use TRSCROSS directly. It does no good to use the segmented files created from HCOPY4/BAS since there is no equivalent restoral program under MS-DOS. To accomplish the port, I threw together the following QnD (quick and dirty) SPLIT program in BASIC which he used to split the database file into two pieces; SPLIT could have broken it into more than two pieces if that was necessary. SPLIT essentially partitions a file into 400 256-byte records or 100K sized chunks. In its simplicity, SPLIT writes the file fragments onto the same drive as the source file; that drive must have sufficient space available. The fragments are named PIECE1, PIECE2, etc...

```
10 INPUT"File to split";NM$
20 OPEN "r",1,NM$
30 FIELD #1,128 AS S1$,128 AS
S2$
40 LAST = LOF(1):COUNT=0:FILE=1
50 FOR I = 1 TO LAST
60 IF COUNT = 400 THEN CLOSE
#2:COUNT = 0: FILE = FILE+1
70 IF COUNT <> 0 THEN 100
80 OPEN "r",2,"piece" +
MID$(STR$(FILE),2):
FIELD #2,128 AS R1$,128 AS R2$
100 GET 1:COUNT=COUNT+1
110 LSET R1$=S1$:R2$=S2$
120 PUT 2
130 NEXT
140 CLOSE
```

Once the file is fragmented, each piece can be copied to a separate floppy disk then TRSCROSS'd to the MS-DOS disk. Once all the pieces are available, the MS-DOS COPY command can be used to concatenate the individual pieces similar to LS-DOS's APPEND command.

Update to XLBOOTC|D/FIX

Fm Frank Slinkman

```
. XLBOOTE/FIX - To use
XLR8er special features
(J.F.R. Slinkman, 22-Feb-
92)
.    Corrects @BANK code
installed by XLBOOTC/FIX
or XLBOOTD/FIX described
in
. article "Final Solution
to the XLR8er Question"
in TMQ 6.1 and included
in
. DiskNotes 6.1.
.    Corrects bug in
jump-to-bank function
which caused SPOOL,
DoubleDuty and
. other routines to fail.
.    Use only on BOOT/SYS
which has ALREADY been
patched by XLBOOTC/FIX or
. XLBOOTD/FIX.
. Apply via PATCH BOOT/
SYS.SYSTEM6 [using]
XLBOOTE/FIX
D06,7E=59;F06,7E=58
D06,81=28;F06,81=27
D06,87=50;F06,87=4F
D06,92=DC;F06,92=DB
D06,A8=E3 C9 3C 67 2E E7
3E 80
F06,A8=E9 3C 67 2E F0 3E
80 07
D06,B0=07 30 01 2D 25 20
F9 26 0D 10 05 2F A6 77
18 15
F06,B0=30 01 2D 25 20 F9
26 08 10 05 2F A6 77 18
15 10
D06,C0=10 05 A6 3E 08 E1
C9 10 07 ED 34 20 F6 B6
18 ED
F06,C0=05 A6 3E 08 E1 C9
10 07 ED 34 20 F6 B6 18
ED 10
D06,D0=10 06 3A 02 02 E1
BF C9 E1 C3 ED 0D E6 03
07 07
F06,D0=06 3A 02 02 E1 BF
C9 E1 C3 ED 0D E6 03 07
07 07
D06,E0=07 07 47 21 78 00
7E E6 8F B0 77 D3 84 AF
C9
F06,E0=07 47 21 78 00 7E
E6 8F B0 77 D3 84 AF C9
00
.eop
```

Model III SuperScripsit

Fm R.Jim Seibert: Dear Roy, I transfered Model III SuperScripsit, (running on a 4) to an LDOS 5.3.0 minimum disk using CONV :3 :1 (Q=N). Everything seems to work fine, including the spell checker which I also transfered, to a different disk, using the same method. However, it will not give a listing of the disk directory when requested from the menu. It will ask for "which drive?", but then not display any files - nothing except the flashing message to press BREAK to get back to the menu. Pressing BREAK sends us back to the menu, which of course still has the option of "going back to TRSDOS".

I loaned a Model III and some software to a refugee family to help the children do their school work. They need all the help they can get to learn our difficult spelling

system. I suppose the loss of the DIR option from within the program is not fatal, but it is a nice feature to have if available. Do you have any ideas? Shouldn't this Dictionary on LDOS 5.3 Data disk also work with SuperScripsit 1.02 working on LSDOS 6.3?

Fm MISOSYS, Inc: Jim, I had your letter and disk sitting on the back burner until I got to issue VI.iv of TMQ; I decided to explore the directory problem in SuperScripsit, first. Since you provided me a copy of your SuperScripsit, and gave a little description of what was happening, it really wasn't too difficult to find the culprit.

I never played with SuperScripsit, but I do know that it uses a series of overlay modules to carry out its work - that's what all of the SCRx/CTL modules are; the printer/CTL modules are the drivers for specific printers. Since I did not know in what module the directory command was located, my first plan of attack was to use FED2 in the disk mode to scan the diskette containing SCRIPSIT/CMD and all the SCRx/CTL modules to look for the string, "which drive?". You threw me a slight curve when you said SS asked "which drive?". That's because I could not find that string. This is just one more reason why it is so important to report exactly what is happening; exactly means verbatim! In any event, I figured it would at least ask using the word "drive". Scanning for that allowed me to wade through the appearances of drive in the SYSTEM modules which were also present on that disk. I finally came to the string, "Which drive do you wish to display (0-3)?". FED2 told me that the string was in SCR17/CTL.

My next step was to feed that control module into my disassembler and locate the code for handling SuperScripsit's directory query. The fragment of code identified as SCR17D/ASM is the specific code which prevents the directory command from working under LDOS. The comments are mine.

```
M8D50 CALL    M7445
      LD      HL,M8D93
;"Which drive...?"
      CALL    M7543
M8D59 CALL    M697D
;Get response
      CP      '0'
      JR      C,M8D59
;Bad if less than "0"
      CP      '4'
      JR      NC,M8D59
;Bad if > 3
      CALL    M7589
      LD      C,00H
      LD      (M4271),A
      CALL    M4419
;@DODIR
      LD      A,0FH
      CALL    M0033
;Cursor off
```

To begin with, note that the fragment restricts the directory query response to the range 0-3. This was fine as far as TRSDOS 1.3 was concerned; however, LDOS supports up to eight drives numbered 0-7.

The key part of the fragment is the CALL to address 4419H. Since this is expected to operate under TRSDOS 1.3, the 4419H address is documented as the TRSDOS 1.3 $DSPDIR service call. According to the TRSDOS 1.3 DOS Manual, the entry conditions state that the program must load address 4271H with the ASCII-coded drive number "0", "1", "2", or "3", then CALL 4419H. At the time of the CALL, register A has your input value - which happens to be in the correct form.

Now under LDOS, the 4419H DOS address is not $DSPDIR but @DODIR, a similar but not exact equivalent service call. Under LDOS, @DODIR entry conditions need the logical drive number in register C and a function code 0-4 in register B (function code 0 is equivalent in operation to $DSPDIR). Thus, even though Tandy supplied patches to SuperScripsit to operate under LDOS - the DOS they used for their Hard Disk Operating System (HDOS), they apparently never provided a patch for this directory command. Since the code has that apparently needless LD C,0 instruction, it gives me the exact quantity of patch bytes to convert the code to LDOS use. What we need to do is convert the value in the accumulator to a logical value (i.e. a binary number) by subtracting 30H from it, then loading the result into register C, and finally loading register B with the function code of 0. So we simply change two of the instructions in the fragment to:

```
LD    B,0
SUB   A,30H
LD    C,A
```

and the fix is in. While I was at it, I also touched up the limit of drive :3 to permit an entry up to drive :7, as well as also touch up the message query. The final patch to SuperScripsit's SCR17/CTL becomes SCR17/FIX which follows. By the way, I don't know if the dictionary data file from the Model III would be usable on the Model 4, but the dictionary program certainly would not be. The two operating system environments are as different as night and day. Unless the program file was actually two programs in one and detected which machine it was running on and adapted its local environment to that, it would require two separate programs. I know of only two programs in the entire TRS-80 market which operate on either machine mode. One is our Gobbling Box program which actually is but a totally self-contained Model III program which automatically switches a Model 4 into a Model III mode if it detects it is running on the Model 4; the other is Anitek's LeScript which is a hybrid-type of program that is actuall a Model III verison and a Model 4 version combined into one file which modifies itself depending on the machine it is running on.

```
. SCR17/FIX Patch to
Model III SS V1.2.08 -
07/31/92
. Corrects Directory
display for LDOS 5.3
. Also allows display of
drives 0-7
. Apply via, PATCH SCR17/
CTL SCR17
D00,36=38:F00,36=34
D00,3C=06 00 D6 30
4F:F00,3C=0E 00 32 71 42
D00,D0=37:F00,D0=33
. Eop
```

Inside TMQ

A new EIA-232 driver for LS-DOS

Four Users on a Four Pea

by James Cameron, Sydney, Australia
>INTERNET:cameronjames@snoc01.enet.dec.com

Why Read This?

This article eventually discusses;

- A Model 4 EIA-232 interface driver,
- A XLR8er ASCI ports interface driver,
- XLR8er Interrupts,
- Input/Output Queueing,
- The XON/XOFF Protocol.

Background

In 1981, while in my eleventh year at school, I came across a game that simulated the construction of galaxy spanning empires, using fleets of ships and involving the colonisation and development of planetary systems. The game was called Starfall, and was published by Yaquinto Publications, Inc., of Dallas Texas.

Each player started out with a planet or two and some space ships. They would go out and "find" other planets to colonise and develop. The goal of the game, apart from having fun, was usually to be the largest empire at the end of a particular turn. Each turn represented 10 years of game time, and each hexagonal section of the universe represented a distance of three light years. The full game of twenty turns often took between 7 and 12 hours, and was thus rarely played.

Part of the fun of the game was the discovery of planets. However, each discovery attempt required about eight double dice rolls and the use of 14 look-up tables to determine the details of the discovery. Not surprisingly, this was the one thing that significantly slowed the game's progress. With one opponent, the solution was to change the rules so that only one discovery attempt could be made each turn. For me, however, automation was the solution.

Initially the software merely automated the discovery procedure, by generating random numbers and looking up the appropriate tables. Slowly, over the years, the software grew to the state it is at now; covering almost 80% of the game's procedures. Players still require the board, playing pieces and associated material; the software just simplifies and speeds up the game.

It has evolved through three languages; BASIC, Assembly, and finally FORTH. The reason for the migration from BASIC to Assembly was that I wanted the opponent in a two player game to be able to use a spare Model I Level I 16k TRS-80 that was lying around. The reason for the migration from Assembly to FORTH was that I wanted to be able to run a multitasking FORTH environment so that up to four players could play at the same time using the one computer. Providing multitasking in BASIC or Assembly language was not as easy as it was in FORTH.

Just to set things straight, this article does not describe the software or the game in any more detail than above. I'm not selling the software; nor giving it away to

anybody, since the game is clearly covered by the appropriate copyright laws. This article describes how I went about setting up things so that four people at once could use the resources of a TRS-80 Model 4p, thus saving many dollars in hardware.

Hardware Inventory

This is the hardware I used to build the four-user setup;

- One TRS-80 Model 4P with XLR8er board,
- Two Tandy 102 Laptop Computers,
- One TRS-80 Model I with expansion interface and EIA-232 card,
- Three cables.

The laptops and the Model I were connected to the Model 4P. I followed the instructions in the XLR8er user manual for installation of the two chips and cables to provide an extra two EIA-232 ports.

The 4P now has three EIA-232 ports. No more will be said on the hardware side of things.

Software

The software was constructed in layers;

(1) an application program written in FORTH,

(2) a multi-tasking FORTH language environment,

(3) serial interface device driver, and;

(4) the operating system; LS-DOS.

The remainder of this article describes the third layer, the device driver for using the three serial interface ports on a TRS-80 Model 4P. The driver program required the following features;

(1) ability to interface with the HD64180's ASCI ports,

```
;++
;     CL3/ASM      V3.0 21-Nov-1991 James Cameron
;     Communications Line Driver for Model 4/4P .
;-
      ORG    3000H
BEGIN JP     GO
;+
;     Relocation Table & Construction Macros
;-
RW_SIZ      DEFL   1024          ; size of relocation table
RW_TAB      DEFS   RW_SIZ
RW_NXT      DEFL   RW_TAB ; next available slot in table
RW    MACRO #ARG=T-2
T     DEFL  $
      ORG   RW_NXT
      DW    #ARG
RW_NXT      DEFL   $
      IFGT  RW_NXT,RW_TAB+RW_SIZ-2
      ERR   ; relocation table overflow, increase allocation
      ENDIF
      ORG   T
      ENDM
RW_B  MACRO #ARG
#ARG  DEFL  RW_NXT
      ENDM
RW_E  MACRO
      RW    0
      ENDM
;+
;     Characters filtered by driver (When not in PASTHRU mode)
;-
XON   DEFL   'Q'-40H       ; Character that will reset PAUSE bit
XOFF  DEFL   'S'-40H       ; Character that will set PAUSE bit
ESCAPE      DEFL   'O'-40H ; Char that will suppress output
BRK   DEFL   'C'-40H       ; Character that will set BREAK bit
FKEY1 DEFL   'F'-40H       ; Flag 1 key for application
FKEY2 DEFL   'N'-40H       ; Flag 2 key for application
DLE   DEFL   0FFH          ; Special buffer control character
;+
;     Supervisor call numbers per Tech Manual
;-
SVC   MACRO #FRODO
      LD    A,#FRODO
      RST   28H
      ENDM
;+
;     Ring buffer macros - Entry - IX=>buffer
;-
RB_PUT      MACRO  #CHAR
      LD    L,(IX+RB_B_INPUT)   ; get input offset
      LD    H,0
      PUSH  IX                  ; point to buffer
      POP   DE
      ADD   HL,DE               ; point to next input byte
      LD    (HL),#CHAR          ; store character
      INC   (IX+RB_B_INPUT)     ; point to next place
      INC   (IX+RB_B_COUNT)     ; kick character count
      LD    A,(IX+RB_B_SIZE)    ; get buffer size
      CP    (IX+RB_B_INPUT)           ; check offset
      JR    NZ,$?
      LD    (IX+RB_B_INPUT),0   ;set new value; rewind to start
$?
      ENDM
RB_GET      MACRO  ; leaves character in C register
```

(2) transmit and receive queueing; up to 256 bytes per port,

(3) XON/XOFF protocol,

(4) Raw data mode,

(5) Full@CTL interfacing (except wakeup support),

(6) Support for installing each port's driver separately,

Each of these features will be discussed below, with subheadings for what it is; why it was needed; and how it was done.

HD64180 ASCI ports

What it is

The XLR8er contains an HD64180 chip, which contains two Asynchronous Serial Communications Interfaces, called ASCI0 and ASCI1. The device driver must be capable of sending and receiving data via these ports.

Why it was needed

To support four users with a minimum of hardware, three ports were required. An alternative solution using some form of ring network and only one port was dismissed due to software complexity, wiring horror and the requirement for ring management intelligence at each remote terminal. Another alternative solution of constructing two extra EIA-232 ports from remains of, say, spare Model III cards, was also skipped.

How it was done

Modules were written in FORTH to prototype access to the ASCI ports using information from the HD64180 manual. These modules worked acceptably, and so were converted into a device driver written in assembly language.

When interrupts were required, a problem

```
      LD    L,(IX+RB_B_OUTPUT)  ; get output offset
      LD    H,0
      PUSH  IX                  ; point to buffer
      POP   DE
      ADD   HL,DE               ; point to next ouput byte
      LD    C,(HL)              ; fetch character
      INC   (IX+RB_B_OUTPUT)    ; point to next place
      DEC   (IX+RB_B_COUNT)     ; kick character count
      LD    A,(IX+RB_B_SIZE)    ; get buffer size
      CP    (IX+RB_B_OUTPUT)    ; check offset
      JR    NZ,$?
      LD    (IX+RB_B_OUTPUT),0  ;set new value; rewind to start
$?
      ENDM
;     Ring Buffer Control Value Offsets
RB_B_SIZE   DEFL  -1      ; size of queue
RB_B_COUNT  DEFL  -2      ; count of characters in queue
RB_B_INPUT  DEFL  -3      ; offset to input end of queue
RB_B_OUTPUT DEFL  -4      ; offset to output end of queue
RB_B_RXTH   DEFL  -5      ; receive buffer xoff threshold
RB_S_BLOCK  DEFL  5       ; size of extra control bytes
;     Code to be executed upon command invokation
GO    PUSH  DE            ; save device control block pointer
      PUSH  HL            ; save command line pointer
      LD    HL,HELLO      ; issue welcome message
      SVC   @DSPLY
      POP   HL
      POP   IX            ; restore dcb pointer
      LD    DE,PARAMS
      SVC   @PARAM
      LD    A,(PA_W_DEBUG)
      OR    A
      JR    Z,GO_1
      SVC   @DEBUG
GO_1  LD    A,(PA_W_P0)
      OR    A
      JP    NZ,P0_DO
      LD    A,(PA_W_P1)
      OR    A
      JP    NZ,P1_DO
      LD    A,(PA_W_CL)
      OR    A
      JP    NZ,CL_DO
      LD    HL,-1         ; error
      RET
PARAMS      DB    80H
      DB    01010000B!2,'CL',0
      DW    PA_W_CL
      DB    01010000B!2,'P0',0
      DW    PA_W_P0
      DB    01010000B!2,'P1',0
      DW    PA_W_P1
      DB    01010000B!5,'DEBUG',0
      DW    PA_W_DEBUG
      DB    0
PA_W_CL           DW    -1    ; default true
PA_W_P0           DW    0     ; default false
PA_W_P1           DW    0     ; default false
PA_W_DEBUG  DW    0     ; default false
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;                 Installation Routines (I_)
;     Prepare the module; grab the @icnfg vector
;     Entry - HL=>link within module
;-
```

```
I_LINK
      SVC    @FLAGS
      LD     A,(IY+28)      ; copy the existing vector to our code
      LD     (HL),A
      INC    HL
      LD     A,(IY+29)
      LD     (HL),A
      INC    HL
      LD     A,(IY+30)
      LD     (HL),A
      RET
I_GET_HILO$                 ; Get address of HILO$ pointer
      LD     DE,'IK'
      SVC    @GTDCB
      DEC    HL
      DEC    HL
      RET
;     Determine address of module in low memory and move it
;     Entry - DE=>module, BC=length, IY=>relocation table
;     Exit  - HL=>new module base, BC=relocation offset
I_MOVE_LOW
      PUSH   DE             ; save base
      CALL   I_GET_HILO$    ; fetch low memory pointer
      LD     E,(HL)
      INC    HL
      LD     D,(HL)
      EX     DE,HL          ; check module will fit
      PUSH   HL             ; old hilo$ value
      ADD    HL,BC          ; calculate end address
      LD     A,H            ; fetch high byte
      CP     13H            ; test for limit
      JR     C,I_MOVE_LOW_1      ; jump if ok
      LD     HL,I_MOVE_ERROR ; issue error message and abort
      SVC    @LOGOT
      SVC    @ABORT
I_MOVE_ERROR
      DB     'Insufficient low memory space',0DH
I_MOVE_LOW_1        ; change hilo$ to protect area
      EX     DE,HL
      LD     (HL),D
      DEC    HL
      LD     (HL),E
      POP    HL             ; base for module
      POP    DE             ; module template
      JR     I_MOVE_COMMON
;     Determine address of module and move it
;     Entry - DE=>module, BC=length, IY=>relocation table
;     Exit  - HL=>new module base, BC=relocation offset
I_MOVE
      PUSH   BC
      LD     B,0            ; fetch high memory value
      LD     HL,0
      SVC    @HIGH$
; calculate start address for module
      POP    BC             ; length
      INC    HL             ; adjust high$
      OR     A              ; clear carry flag
      SBC    HL,BC          ; calculate new start address
; update the system's high memory pointer
      PUSH   HL             ; new base
      PUSH   BC             ; length
      DEC    HL             ; point to free memory ending address
      LD     B,0            ; Request change of HIGH$
      SVC    @HIGH$
```

was found. Since HD64180 interrupts are delivered using a vector table, the table had to be positioned somewhere. Since an interrupt could occur at any time, code had to be borrowed to ensure that the appropriate bank is switched in so that the device driver interrupt could be called. The vector table and the bank switching code associated with it had to be installed in low memory.

Queueing

What it is

When characters are to be transmitted, they are placed in an output queue and the driver returns very quickly to the application program. When the serial interface is ready to transmit, it triggers an interrupt routine within the driver (transmit register empty interrupt) which takes a character from the output queue and sends it.

When a character is received by the serial interface, it triggers another interrupt routine within the driver (receive register full interrupt), which accepts the character from the interface and places it at the end of the input queue. When the application program is awaiting input, it calls the driver asking for a character. If the input queue is empty, then nothing is returned. If the input queue has at least one character in it, that character is removed from the queue and returned.

Why it was needed

Any number of keystrokes may arrive without the application program being able to handle them in time. Having the driver perform input queueing means that the application program is greatly simplified.

With four simultaneous users, response time becomes an issue. With queueing, the transmission of a line of text from the application program will be rapid, even though the actual transmission from the output queue to the remote device will be

```
        POP     BC              ; length
        POP     HL              ; new base
;       I_MOVE_COMMON, move module into place
;       Entry - BC=length, DE=module, HL=destination
I_MOVE_COMMON
        EX      DE,HL
        PUSH    HL              ; from address
        PUSH    DE              ; to address
        LDIR                    ; do move
        POP     HL              ; restore new module address
        POP     DE              ; restore template address
        PUSH    HL              ; save new module address
        OR      A               ; calculate relocation offset
        SBC     HL,DE
; perform relocation address fixup
; hl=offset to add to each reference in module
; de=module template start address
        LD      C,L             ; prepare for relocation loop
        LD      B,H
RW_LOOP
        LD      L,(IY)          ; fetch relocation pointer
        LD      H,(IY+1)
        LD      A,H
        OR      L
        JR      Z,RW_END        ; found end of table
        LD      E,(HL)          ; get value from template
        INC     HL
        LD      D,(HL)
        DEC     HL
        EX      DE,HL           ; change the value by the offset
        ADD     HL,BC
        EX      DE,HL           ; change the pointer too
        ADD     HL,BC
        LD      (HL),E          ; store the new reference
        INC     HL
        LD      (HL),D
        INC     IY              ; move to next table entry
        INC     IY
        JR      RW_LOOP
RW_END  POP     HL              ; new module base
        RET
;Set up DCB - Entry - HL=>driver code, IX=>DCB
I_DCB   RES     3,(IX)          ; Turn off NIL flag
        SET     2,(IX)          ; Enable @CTL requests
        SET     1,(IX)          ; Enable @PUT requests
        SET     0,(IX)          ; Enable @GET requests
        LD      (IX+1),L        ; Set driver vector in DCB
        LD      (IX+2),H
        RET
;       Insert initialisation routine into @ICNFG chain
;       Entry - DE=>driver initialisation routine (original)
I_INIT  EX      DE,HL
        PUSH    DE
        SVC     @FLAGS
        ADD     HL,BC           ; offset to memory location
        LD      (IY+29),L
        LD      (IY+30),H
        LD      A,0C3H
        LD      (IY+28),A
        LD      DE,I_INIT_1     ;       Initialise interface
        PUSH    DE
        JP      (HL)            ; call initialisation routine
I_INIT_1 POP    HL              ; restore saved registers
        RET
```

slow. This frees up the application program to handle requests from the other users. Without queueing, the whole application seems rather jerky.

Also, in cases where large amounts of data (such as programs) are being transmitted, an output queue allows more efficient transmission. When downloading a program to a Tandy 102, with XON/XOFF protocol, the LS-DOS COPY command works fine. The queueing means that the transmission is sustained without any breaks caused by disk access. The COPY command fills the output queue; and when it has to perform disk I/O, the transmission keeps going using the contents of the queue.

How it was done

The driver contains two 256 byte queues; one for transmit and one for receive.

The input end of the transmit queue is connected to the @PUT service request in the driver. The @PUT will stall if the queue is full. If, when the @PUT completes, the queue has exactly one character in it, then that means that the transmit register empty interrupt will have to be enabled.

The output end of the transmit queue is handled by the transmit register empty interrupt routine. When an interrupt occurs, the next character is taken from the queue and placed into the transmit register. This clears the interrupt. If no character is available (queue empty), then the interrupt must be cleared by disabling it.

The input end of the receive queue is handled by the receive register full interrupt routine. When an interrupt occurs, the received character is taken from the receive register and placed into the queue.

The output end of the receive queue is connected to the @GET service request in the driver. The @GET checks to see if the queue contains any characters. If the queue is empty, then the flags are set

accordingly and the routine exits. Otherwise, the next character is returned.

XON/XOFF Protocol

What it is

XOFF means "hang on for a moment". XON means "ok, continue". XOFF is an abbreviation for Transmit Off; XON is an abbreviation for Transmit On.

Imagine one machine sending data to another; the recipient runs out of memory or something, and has to ask the sending machine to stop for a while so that it can save the data to disk. It sends an XOFF back to the sending machine, which (hopefully), recognises the XOFF as a request to stop transmitting. When the receiving machine is ready for more data, it sends an XON. The sending machine starts up again.

The most popular coded values for XON and XOFF are the equivalent of Control/Q and Control/S; or DC1 and DC3, or hexadecimal 11 and 13, or decimal 17 and 19. Machines that support the XON/XOFF protocol usually recognise these characters as requests to start or stop transmission.

Why it was needed

The devices connected to the central machine are capable of using this protocol to allow high speed transmission and yet prevent input queue overrun.

How it was done

Software that communicates via serial interfaces is typically divided into two parts. The "driver" program is installed and connected to the operating system, and does the actual work of moving data between memory and the serial interface. The "application" program is the one that is using the services of the "driver". Application programs can be written in almost any language; examples are

```
;     Confirm installation; show module base address to user
;     Entry - H&L=>new module installed
I_CONFIRM   PUSH   HL
      EX    DE,HL          ; store module address in message
      LD    HL,MSG_2
      SVC   @HEX16
      LD    HL,MSG_0       ; issue first part of message
      SVC   @DSPLY
      POP   HL             ; display module name
      PUSH  HL
      INC   HL             ; point to length byte
      INC   HL
      INC   HL
      INC   HL
      LD    B,(HL)         ; get length
      INC   HL
I_CONFIRM_1 LD    C,(HL)         ; get character
      INC   HL
      SVC   @DSP
      DJNZ  I_CONFIRM_1
      LD    HL,MSG_1
      SVC   @DSPLY
      POP   HL
      RET
HELLODB     'CL T3.0 Communications Line driver, '
      DB    'Copyright (c) 1992 James Cameron',0AH,0DH
MSG_0DB     'Module ',3
MSG_1DB     ' installed at X',27H
MSG_2DB     '0000',27H,0DH
      DB    'Configured; SETCOM (BAUD=4800,WORD=8,STOP=2,'
      DB    'PARITY=NONE,DTR,RTS)',0DH
*GET CL
*GET HITACHI
*GET P0
*GET P1
*GET IN
      END   BEGIN
```

```
;++
;     CL/ASM
;+
;     Addresses used within V6.2 of TRSDOS
;-
CL_B_SIEM   DEFL   003DH  ; Software interrupt enable mask
CL_B_HIEM   DEFL   0080H  ; $FLAGS + 22 (WRINTMASK$)
CL_W_RXI    DEFL   0048H  ; Vector for UART receive interrupt
CL_W_TXI    DEFL   0046H  ; Vector for UART transmit interrupt
;+
;     Hardware addresses
;     Bit numbers for WRINTMASK$ (Interrupt enable mask)
;-
CL_C_TXI    DEFL   4      ; enable transmit buffer empty
CL_C_RXI    DEFL   5      ; enable receive buffer full
;+
;     Port numbers
;-
CL_P_HIEM   DEFL   0E0H          ; Interrupt enable mask
CL_P_BRG    DEFL   0E9H          ; UART Baud Rate Register
CL_P_STATUS DEFL   0EAH          ; UART Status Register
CL_P_DATA   DEFL   0EBH          ; UART Data Register
;+
;     Module installation
```

```
;-
CL_DOLD     HL,CL_LINK
      CALL  I_LINK        ; grab a copy of the @icnfg vector
      LD    DE,CL_BASE    ; module start
      LD    BC,CL_LENGTH  ; module length
      LD    IY,CL_RW      ; module relocation table
      CALL  I_MOVE        ; move module to system memory
; hl=module installed address
; bc=relocation offset
      CALL  I_DCB         ; set up dcb
      CALL  CL_INSTALL    ; set system vectors
      LD    DE,CL_INIT
      CALL  I_INIT        ; install and invoke initialisation
      CALL  I_CONFIRM     ; confirm installation
      LD    HL,0
      RET
;+
;     Initialise operating system vectors for interrupts
;     Entry - BC=relocation offset
;-
CL_INSTALL
      PUSH  HL
      DI
      LD    HL,TX         ; Point to transmit interrupt routine
      ADD   HL,BC
      LD    (CL_W_TXI),HL; Tell DOS where to go
      LD    HL,RX
      ADD   HL,BC
      LD    (CL_W_RXI),HL
      LD    HL,CL_B_SIEM ; Pt to software interrupt enable mask
      SET   CL_C_TXI,(HL); Enable transmit interrupts
      SET   CL_C_RXI,(HL); Enable receive interrupts
      LD    HL,CL_B_HIEM ; Pt to hardware interrupt enable mask
      SET   CL_C_TXI,(HL); Enable transmit interrupts
      SET   CL_C_RXI,(HL); Enable receive interrupts
      LD    A,(HL)        ; Get the mask and
      OUT   (CL_P_HIEM),A; Output it to hardware
      EI
      POP   HL
      RET
;+
;     Driver header and code
;-
      RW_B  CL_RW
CL_BASE   JR    CL_START
      DW    CL_LAST
      RW
      DB    3,'_CL'             ; Module name text
      DEFW  0             ; DCB pointer
      DEFW  0             ; TRSDOS reserved
;+
;     Flag bits used by driver
;-
FLAGSDEFB   0
CONFIG_0    DB    0CCH    ; baud=4800
CONFIG_1    DB    0FFH    ;
   parity=none,word=8,stop=2,dtr,rts
;+
;     Bit definitions within FLAGS
;-
PAUSEDEFL   0       ; Output paused due to XON received
PASS DEFL   1       ; In PASTHRU mode, no filtering done
SUPP DEFL   2       ; Output being suppressed at bequest of
   user
```

KERMIT, COMM/CMD, terminal emulators, and specialised programs for device control; such as model railways.

Now, a question that arises when implementing a system that requires XON/XOFF protocol is where to put it. Do we put it in the application program or do we put it in the driver program that is common across all applications? In LS-DOS, the position taken is to make flow control the responsibility of the application program. This saves memory, because the extra program code required for flow control is not installed.

Ideally, however, the best place for such protocol logic is within the driver program. This is the approach used in this driver. The advantages are; (1) the logic does not need to be implemented in every application program, (2) the application program does not have to execute fast enough to stop or start transmission, and (3) if the protocol changes, (e.g. to a byte-count oriented packet-with-checksum protocol such as DDCMP) the application program(s) need not be changed. The disadvantages include; (1) higher usage of system memory, (2) increased build and maintenance cost of the driver program (it's no longer a simple task), (3) possible increased usage of processor time to handle each character. TANSTAAFL - There Ain't No Such Thing As A Free Lunch.

To implement XON/XOFF is reasonably simple. The driver must keep a flag which indicates whether transmission has been paused or not. When an XOFF is received, the pause flag is set. When an XON is received, the pause flag is cleared. The XON/XOFF character that is received is not stored for the application program. The transmit register empty interrupt routine checks the flag prior to attempting transmission. If the flag is cleared, then transmission proceeds normally. If the flag is set, then the transmit register empty interrupt is disabled and the routine exits. The XON receive routine also enables the transmit register empty interrupt.

Raw data mode (PASTHRU)

What it is

Raw data mode, or PASTHRU as it is called in the source code, is simply a flag that is set to indicate that all data must pass through the driver without any translation or filtering.

Why it was needed

When XON/XOFF is implemented, transmission of binary data will suffer, since any occurrence of the XON or XOFF characters will be misapplied. PASTHRU mode effectively turns off the XON/XOFF protocol. Binary transfers are used for loading memory images from the Tandy 102 into the 4P.

How it was done

A flag in the driver represents the PASTHRU state. A @CTL request can be used to turn it on or off. When it is on, incoming XON or XOFF characters are ignored and placed into the input queue.

@CTL Interfacing

What it is

Application programs or the operating system send requests to device drivers to perform particular tasks. Such requests are called Control requests, and they use the @CTL supervisor call. The TRS-80 Model 4/4P Technical Reference Manual (26-2119) on page 42 and 65 describes the @CTL interface to device drivers. The Control Code Implementation Table shows the @CTL function values (requests) and whether they are implemented by the driver;

```
BREAKDEFL  3       ; interrupt character detected
FLAG1DEFL  4       ; Flag for flag key 1
FLAG2DEFL  5       ; Flag for flag key 2
RXOR DEFL  6       ; receive buffer became full
CL_INIT                    ; Set UART parameters

     LD    HL,FLAGS
     RW
     CALL  CL_CTL_02
     RW
CL_LINK    RET             ; @ICNFG Vector
     NOP
     NOP
CL_START                   ; Driver code executed by call to
                           ; SVC's @GET, @PUT, and @CTL
     LD    HL,FLAGS
     RW
     JP    C,CL_GET        ; Service a @GET call
     RW
     JP    Z,CL_PUT        ; Service a @PUT call
     RW
CL_CTL     LD    A,C       ; Get the control code
     OR    A               ; Test for zero - status request
     JR    Z,CL_CTL_00     ; Jump if so
     DEC   A               ; test for 1
     DEC   A               ; test for 2 - initialise driver
     JR    Z,CL_CTL_02
     DEC   A               ; test for 3 - clear buffers
     JR    Z,CL_CTL_03
     LD    A,C
     CP    0F8H            ; Test for break detection
     JR    Z,CL_CTL_F8
     CP    0FCH            ; test for suppression detect/reset
     JR    Z,CL_CTL_FC     ; Jump if so
     CP    0F9H            ; Test for set PASTHRU
     JR    Z,CL_CTL_F9     ; Jump if so
     CP    0FAH            ; Test for reset PASTHRU
     JR    Z,CL_CTL_FA
     CP    0E0H            ; Test for flag 1 detection
     JR    Z,CL_CTL_E0     ; Jump if so
     CP    0E1H            ; Test for flag 2 detection
     JR    Z,CL_CTL_E1     ; Jump if so
     XOR   A               ; Normal status return
     RET
CL_CTL_00                  ; Test device status
     LD    A,(CL_TXB+RB_B_COUNT)      ; Get output buffer
  byte counter
     RW
     OR    A      ; Set flags, ready if no characters
     IN    A,(CL_P_STATUS) ; Return status register of UART
     RET
CL_CTL_02  ; driver initialisation
     LD    A,(CONFIG_0)
     RW
     OUT   (CL_P_BRG),A ; set baud rate
     LD    A,(CONFIG_1)
     RW
     OUT   (CL_P_STATUS),A
     LD    (HL),0          ; clear flag bits
CL_CTL_03  ; clear buffers
     RES   PAUSE,(HL)
     RES   SUPP,(HL)
     RES   RXOR,(HL)
     DI                    ; Disable interrupts
```

```
        XOR     A
        PUSH    IY
        LD      IY,CL_RXB
        RW
        LD      (IY+RB_B_COUNT),A
        LD      (IY+RB_B_INPUT),A
        LD      (IY+RB_B_OUTPUT),A
        LD      IY,CL_TXB
        RW
        LD      (IY+RB_B_COUNT),A
        LD      (IY+RB_B_INPUT),A
        LD      (IY+RB_B_OUTPUT),A
        POP     IY
        CALL    CL_ITX
        RW
        EI
        RET
CL_CTL_FC
        LD      IX,CL_TXB
        RW
        LD      C,DLE           ; Place a DLE and a zero into buffer
        CALL    CL_PUT_0
        RW
        XOR     A
        LD      C,A
        CALL    CL_PUT_0
        RW
        XOR     A
        RET
CL_CTL_F8
        LD      A,(HL)          ; Get flags
        RES     BREAK,(HL)      ; Turn off break bit for next time
        BIT     BREAK,A         ; Test old break bit
        RET             ; Return flag set, NZ=Break Detected
CL_CTL_F9
        LD      A,(HL)
        SET     PASS,(HL)       ; Set PASTHRU mode
        BIT     PASS,A          ; Return previous state
        RET
CL_CTL_FA
        LD      A,(HL)
        RES     PASS,(HL)       ; Reset PASTHRU mode
        BIT     PASS,A          ; Return previous state
        RET
CL_CTL_E0
        LD      A,(HL)
        RES     FLAG1,(HL)      ; Turn off this flag
        BIT     FLAG1,A         ; Return old state of flag, NZ=True
        RET
CL_CTL_E1
        LD      A,(HL)
```

Control Code Implementation Table

Code	Purpose	Implemented	Used By
00H	Test device status	y	SPOOL
01H	Send BREAK or force interrupt	n	COMM/CMD
02H	Initialise	y	@ICNFG
03H	Clear buffers	y	
04H	Set wakeup vector	n	COMM/CMD
08H	Preview next receive character	n	

Why it was needed

The application needed to be able to clear the queues and enable or disable PASTHRU mode. The best way to provide this functionality was via the @CTL interface.

How it was done

The status, initialise, and clear queues requests were implemented in the driver. The wakeup vector was not required; but it may be added later so that COMM/CMD can be used.

Two new @CTL function codes were implemented to turn on or off the PASTHRU flag.

Separate Driver Modules

What it is

There is one driver module for each port, and an additional interrupt dispatcher module required for use of the HD64180 ports. Each module can be installed separately.

Why it was needed

Not all applications will require all three driver modules to be installed, and it is a waste of memory to install more than is required. Some applications may not run with a HD64180 processor, and so the HD64180 port specific code is not required. Also, the operating system command syntax does not support the creation of more than one device at a time.

How it was done

The entire set of three drivers is packaged within one driver installation program which is used by the LS-DOS SET command. The installation program examines command line parameters to determine which modules should be installed. To install all three driver programs, three LS-DOS commands are required;

```
SET *CL CL3/DVR
SET *P0 CL3/DVR (P0)
SET *P1 CL3/DVR (P1)
```

Source Modules

There are five distinct source modules that are used to assemble the final installation program;

- CL3/ASM Main program,
- CL/ASM Model 4 standard EIA-232 interface,
- P0/ASM HD64180 ASCI channel 0,
- P1/ASM HD64180 ASCI channel 1,
- IN/ASM HD64180 Vectored Interrupts,
- HITACHI/ASM HD64180 Instructions.

CL3/ASM

This module is the main program module, and *GETs the required modules during assembly. Execution starts here as a result of a SET command that creates a device. The program checks the parameters, if any, and calls the appropriate module installation routine. Subroutines that are called by the module installation routines are within this module.

CL/ASM, P0/ASM, P1/ASM

These modules contain the interface code for each port. Each module contains the following sections; installation routine, driver module header, initialisation and @ICNFG routine, @CTL handler, @PUT handler, @GET handler, interrupt handler, buffers and control blocks.

IN/ASM

This module contains the HD64180 Vec-

```
      RES   FLAG2,(HL)
      BIT   FLAG2,A
      RET
CL_PUT
      LD    IX,CL_TXB       ; point to transmit control block
      RW
      LD    A,C             ; Get character to put
      CP    DLE             ; Test for DLE
      CALL  Z,CL_PUT_0      ; Send the DLE twice
      RW
CL_PUT_0
      LD    A,(IX+RB_B_COUNT)   ; Get count of pending chars
      CP    0FEH            ; Test for upper limit
      JR    NC,CL_PUT_0     ; Loop until buffer a bit more empty
      DI
      RB_PUT C
      LD    A,(IX+RB_B_COUNT)   ; get character count
      DEC   A               ; Test for 1 byte in buffer
      CALL  Z,CL_ITX        ; Start transmit interrupt routine
      RW
      EI
      XOR   A               ; Set return status
      RET
CL_GET
      LD    IX,CL_RXB
      RW
      LD    A,(IX+RB_B_COUNT)   ; count of waiting characters
      OR    A               ; Test it
      JR    Z,CL_GET_1      ; Jump if none waiting
      DI                    ; Disable interrupts
      RB_GET
      LD    A,C
      EI
      CP    A               ; Set zero flag to indicate
      RET                   ; Byte received and RETURN
CL_GET_1
      OR    1               ; Return zero with NZ flag
      LD    A,0
      RET                   ; To say no character found
;     RX, routine to handle interrupt by UART upon character
;     received from remote device.
RX    DEFL  $
      IN    A,(CL_P_STATUS)     ; Get UART status register
      BIT   7,A             ; data received?
      RET   Z               ; no! this isn't our interrupt!
      LD    HL,FLAGS        ; Point to flags byte
      RW
; needs fixing here, character should only be accepted if
; UART says it's error free, and also only if line is still
; connected, otherwise it's garbage data
      IN    A,(CL_P_DATA); Get UART received data register
      BIT   PASS,(HL)       ; Test PASTHRU flag
      JR    NZ,RXS5         ; If set then don't test character
      CP    XOFF            ; Test for XOFF request
      JR    NZ,RXS1               ; If not then jump ahead
      SET   PAUSE,(HL)      ; Set the pause flag
      RET
RXS1  CP    XON             ; Test for XON request
      JR    NZ,RXS2               ; If not then jump ahead
      RES   PAUSE,(HL)      ; Reset the pause flag
      JP    CL_ITX          ; Initialise transmission
      RW
RXS2  CP    ESCAPE          ; Test for ESCAPE request
      JR    NZ,RXS3               ; If not then jump ahead
```

```
        BIT     SUPP,(HL)       ; Test suppression flag
        JR      NZ,RXS4                 ; Is set, so go and reset it
        SET     SUPP,(HL)       ; Turn on suppression
        RET
RXS4    RES     SUPP,(HL)       ; Turn off suppression
        JP      CL_ITX          ; Initialise transmission
        RW
RXS3    CP      BRK             ; Test for BREAK key request
        JR      NZ,RXS6
        SET     BREAK,(HL)      ; Set break bit in flags
        RET
RXS6    CP      FKEY1           ; Test for flag key 1
        JR      NZ,RXS7                 ; Jump if not
        SET     FLAG1,(HL)      ; Set flag bit
        RET
RXS7    CP      FKEY2           ; Test for flag key 2
        JR      NZ,RXS5                 ; Jump if not
        SET     FLAG2,(HL)      ; Set flag bit
        RET
;       Place the incoming character into the receiver buffer
;       (Note that characters received when the buffer is full
;       will cause a buffer wrap-around and loss of one buffer's
;       worth of data)
RXS5    DEFL    $               ; Insert character into buffer
        PUSH    IX
        LD      IX,CL_RXB
        RW
        LD      C,A
        LD      A,(IX+RB_B_COUNT)
;       CP      (IX+RB_B_RXTH)          ; is buffer getting full?
;       CALL    NC,RXS8                 ; send an xoff if not yet done
        INC     A               ; is it about to hit the roof?
        JR      Z,RXS9          ; yes; handle overrun
        RB_PUT C
        POP     IX
        RET
RXS9                            ; rx buffer overrun
        SET     RXOR,(HL)       ; set flag
        POP     IX
        RET
;TX, routine to handle interrupt by UART upon character
;fully transmitted to remote device. i.e. transmit buffer
;is now empty.
TX      DEFL    $
        IN      A,(CL_P_STATUS)         ; Get UART status register
        BIT     6,A             ; Test transmitter holding register
        RET     Z               ; If NOT empty then ignore
        LD      A,(CL_TXB+RB_B_COUNT)       ; Get counter
        RW
        OR      A               ; Test it
        JR      Z,TXS2          ; Buffer is empty, jump
        LD      A,(FLAGS)       ; Get flag bits
        RW
        BIT     PAUSE,A                 ; Test pause bit
        JR      NZ,TXS2                 ; Pause mode on, jump
        BIT     SUPP,A          ; Test suppression bit
        JR      NZ,TXS3                 ; Suppression on, so jump
TXS4    CALL    TXG             ; Dequeue a character from buffer
        RW
        CP      DLE             ; Test for Data Link Escape
        JR      Z,TXDLE                 ; Yes, so jump
        OUT     (CL_P_DATA),A;  Send character
        RET
TXDLE   CALL    TXG             ; Dequeue command character
```

tored Interrupt module, which is placed in low memory to handle HD64180 generated interrupts. It is only installed if required, and only if it is not already installed. The module ensures that bank 0 is enabled before the port interface driver interrupt routine is called.

HITACHI/ASM

This module contains macros for generating HD64180 specific instructions.

TRS-80 Serial Interface Comparison Table

The ***Features Table*** compares the various features and configurations supported by the EIA-232 interfaces available to me.

(1) Although these interrupts are implemented in hardware, no software has been seen that uses them.

(2) Not only is DCD available on ASCI0, but **it must be active before transmission is possible**. There is a hardware interlock. I found that a continuous receive interrupt condition also exists when DCD is inactive. This interrupt could not be cleared except by disabling it or making DCD active. The HD64180 manual both supports and denies this, according to which page you read - the logic diagram is correct but the prose isn't.

(3) No opportunity to test this; but it is probably present.

(4) Although these lines are unavailable for monitoring by the HD64180, a competent hardware hacker would just rewire the interface plug. In the writer's view, the Tandy interface is a case of overkill.

(5) Who needs 3600 baud? Good question; but note that the Tandy interfaces supports some pretty strange baud rates (50, 110, 134.5, 1800, 2000, 3600, and 7200), whereas the HD64180 only supports baud rates derived by repeated halving of 38400 baud. [-editor's note: actually 110, and 134.5 baud were quite com-

mon in teletypewriter private line and switched dial-up (TWX) communications back in the 60's and late 70's; 50 baud was, I believe, used in the old teletypesetters - back in the days before photo-composers]

The End

I've played one full game and three small ones using this setup. Of course, I've found areas in my game software that would benefit from a bit of fun - er, recoding. The other players found the interactive response to be fine; except when disk I/O was in progress - my FORTH environment goes into a single-threaded mode when that happens. I'll fix that someday; maybe by adding a deferred write-back disk sector cache.

So now I have a machine that can easily provide service to four people at the same time. You don't need a Mess DOS machine or an 80n86 processor to do multitasking; just some intelligent software.

```
        RW
        CP      DLE             ; Test for another DLE
        JR      NZ,TXCOM        ; No, jump to command
        OUT     (CL_P_DATA),A   ; Send character
        RET
TXCOM   JR      TXDONE          ; Ignore it for the moment
; if it was a suppression reset we don't need to do anything,
; if it wasn't then we don't know what to do with it anyhow.
TXG     DEFL    $
        PUSH    IX
        LD      IX,CL_TXB
        RW
        RB_GET
        LD      A,C
        POP     IX
        RET                     ; Return character
;       Suppression mode detected, so look for a
;       suppression mode reset sequence.
TXS3    LD      A,(CL_TXB+RB_B_COUNT)
        RW
        LD      B,A             ; Set character counter
TXS3L   CALL    TXG             ; Get character in buffer
        CP      DLE             ; Check for DLE
        JR      Z,TXS3D         ; Go if so
TXS3L1
        DJNZ    TXS3L           ; Loop until buffer empty
        JR      TXS2            ; Completed
TXS3D   DEC     B
        CALL    TXG             ; Get character after DLE
        CP      DLE             ; Test for a real DLE
        JR      NZ,TXS3C        ; No, so it's a command
```

CL3: Additional Remarks from MISOSYS

by Roy Soltoff

James Cameron has put together quite an interesting project.. It shows you just what can be accomplished with the *old* computer hardware providing the right amount of initiative and investigative work is carried out. On the other hand, another XLR8er user would not be able to easily use the provided driver as is. I consider myself somewhat experienced, yet it took me a few days of playing, conniving, and testing to get everything in the CL3/DVR driver to work. The problem is in understanding some of the finer aspects of how the ASCI ports in the 64180 processor used in the XLR8er board behave and what you have to do from a signalling standpoint to interface to the ports at the RS-232 level. I fell into just about every trap, and eventually extricated myself. These additional remarks should enable

FEATURES TABLE

HD64180 Feature	HD64180 Model I	Model IV	ASCI0	ASCI1
Dip Switches	Y	N	N	N
Min Baud	50	50	37	37
Max Baud	19200	19200	38400	38400
Bits per character	5-8	5-8	7-8	7-8
Split Speed	Y	Y	N	N
Receive Interrupt	N	Y	Y	Y
Transmit Interrupt	N	Y(1)	Y	Y
Error Interrupt	N	Y(1)	Y	Y
DMA Mode	N	N	Y	Y
CTS	Y	Y	Y	Y
DSR	Y	Y	N(4)	N(4)
DCD	Y	Y	Y(2)	N
RI	Y	Y	N(4)	N(4)
RD input bit	Y	(3)	N	N
DTR	Y	Y	N(4)	N(4)
RTS	Y	Y	Y	N
3600 Baud (5)	Y	Y	N	N

For notes (see text)

```
      JR     TXS3L1
TXS3CDEFL    $              ; Found a DLE followed by something
                            ; other than a DLE, assume it's a
                            ; suppression reset request.
      LD     HL,FLAGS       ; Point to flags
      RW
      RES    SUPP,(HL)      ; Reset Suppression flag
      JR     TXS4           ; Go and get character to send
TXS2  DEFL   $              ; No need to send more characters
                            ; until something else happens, so
                            ; disable this interrupt...
      LD     HL,CL_B_HIEM ; Point to interrupt enable mask
      RES    CL_C_TXI,(HL); Reset transmission interrupt enable
      LD     A,(HL)         ; Get mask and
      OUT    (CL_P_HIEM),A; Output it to hardware
TXDONE       DEFL   $
      RET
;CL_ITX, Initialise transmission - enables interrupt for
;UART transmit register empty, causing the TX interrupt routine
;to be called.
CL_ITX       DEFL   $
      LD     HL,CL_B_HIEM ; Point to interrupt enable mask
      SET    CL_C_TXI,(HL); Enable transmit interrupts
      LD     A,(HL)         ; Get mask and output it
      OUT    (CL_P_HIEM),A; to hardware
      RET
      RW_E
;     Receive and transmit buffers, 256 bytes each
CL_TXS       DEFL   256
CL_RXS       DEFL   256
      DC     RB_S_BLOCK,0
CL_RXB       DC     CL_RXS,0
      DC     RB_S_BLOCK,0
CL_TXB       DC     CL_TXS,0
;
CL_LAST            DEFL   $-1
CL_LENGTH    DEFL   $-CL_BASE     ; total length of module
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;     P0/ASM
P0_DOCALL    IN_WHERE       ; find IN module
      RET    NZ             ; failed to find it
      PUSH   HL     ; save pointer to IN module
      LD     HL,P0_LINK
      CALL   I_LINK         ; grab @icnfg
      LD     DE,P0_BASE     ; module start
      LD     BC,P0_LENGTH ; module length
      LD     IY,P0_RW       ; module relo table
      CALL   I_MOVE         ; move to sys mem
      CALL   I_DCB          ; set up dcb
      POP    DE     ; restore IN module ptr
      CALL   P0_INSTALL    ; install P0 module
      LD     DE,P0_INIT
      CALL   I_INIT ; install and invoke initialisation
      CALL   I_CONFIRM      ; confirm install
      LD     HL,0
      RET
P0_INSTALL   PUSH   BC
      PUSH   HL
      LD     HL,P0_INTERRUPT      ; our interrupt handler
      ADD    HL,BC  ; converted to actual installed address
      EX     DE,HL
      LD     BC,P0_IV-IN_BASE     ; offset within IN module
```

other XLR8er users to more easily adapt the CL3 driver for their own purposes.

But first, I want to note that James and I had sent some electronic mail messages discussing his intended use of the driver. I wrote John that I didn't think he would be able to get away so easy from not commenting further on the Starfall game. I suspected that TMQ readers would find that game of significant interest. But his response to that subject tells the tale.

Concerning the Starfall game, he wrote "I agree. But you see my problem? The software was developed for personal use and requires the presence of the actual game hardware (playing board, rulebook and counters). I don't know if it's still available, or if the company still exists. Short of actually clearing it with them, I'll just try to stay low. Also, the software is by no means complete. There are gaping holes in the simulation, which are handled by manual procedures. I'd want to make it a quality product long before I'd consider making it available to others. I can't offer anyone the game software until I am sure of the legality, but the actual FORTH multitasking environment is mine to give away."

I wrote James that I would keep him posted on the feedback I get from readers. His response, "They are welcome to contact me directly. My internet address is below, and my snail-mail address is P.O. Box 339, Epping, 2121, Australia."

From INTERNET: (cameronjames @snoc01.enet.dec.com)

From CompuServe: >INTERNET: cameronjames@snoc01.enet.dec.com

Now in connection with expanding on the CL3 article, first note that this driver actually includes three separate drivers. One is a replacement for the standard Model 4 COM/DVR - the driver used to access the Model 4's Serial Input/Output (SIO) hardware. However, it is not an exact replacement. Although it supports send and receive 256-byte character buffer, it is not written to be able to utilize

```
        ADD     HL,BC           ; point to vector
        LD      (HL),E          ; store our
        INC     HL              ; interrupt
        LD      (HL),D ; handler
        POP     HL              ; address
        POP     BC
        RET
        RW_B    P0_RW
P0_BASE JR      P0_START
        DW      P0_LAST
        RW
        DB      3,'_P0'
        DW      0,0             ; reqd for DOS
P0_P_CA DEFL    0       ; control register a
P1_P_CA DEFL    1       ; control register a
CA_M_MOD0 DEFL  1H      ; stop bits, 0=1, 1=2
CA_M_MOD1 DEFL  02H     ; parity, 1=enabled
CA_M_MOD2 DEFL  04H     ; bits per char
CA_M_MPBREFR    DEFL    08H     ; multi-
; processor bit receive & error flag reset
CA_M_RTS0 DEFL  10H     ; RTS channel 0
CA_M_CKA1D DEFL 10H     ; cka1 clock disable
CA_M_TE DEFL    20H     ; transmitter enable
CA_M_RE DEFL    40H     ; receiver enable
CA_M_MPE DEFL   80H     ; multi CPU enable
P0_P_CB DEFL    02H     ; control register b
P1_P_CB DEFL    03H     ; control register b
CB_M_SS0 DEFL   01H     ; divide ratio
CB_M_SS1 DEFL   02H
CB_M_SS2 DEFL   04H
CB_M_DR DEFL    08H     ; divide ratio
CB_M_PEO DEFL   10H     ; parity even/odd
CB_M_CTSPS DEFL 20H     ;CTS / prescale
CB_M_MP DEFL    40H     ;multi CPU mode
CB_M_MPBT DEFL  80H     ;multi CPU bit xmit
P0_P_STATUS DEFL 04H    ; status register
P1_P_STATUS DEFL 05H
ST_M_TIE DEFL   01H     ; xmit int enable
ST_M_TDRE DEFL  02H     ; xmit DR empty
ST_M_DCD0 DEFL  04H     ; data CD
ST_M_CTS1E DEFL 04H     ; 1 = if cts else
   rxs then
ST_M_RIE DEFL   08H     ; recv int enable
ST_M_FE DEFL    10H     ; framing error
ST_M_PE DEFL    20H     ; parity error
ST_M_OVRN DEFL  40H     ; overrun error
ST_M_RDRF DEFL  80H     ; receive DR full
P0_P_TDR DEFL   06H     ;xmit DR
P1_P_TDR DEFL   07H
P0_P_RDR DEFL   08H     ; recv DR
P1_P_RDR DEFL   09H
P0_FLAGS DEFB   0
P0_INIT ;Driver initialisation routine
        LD      HL,P0_FLAGS
        RW
        CALL    P0_CTL_02
        RW
P0_LINK RET     ;       @ICNFG Vector
        DW      0
P0_START ; Driver code executed by call to
        ; SVC's @GET, @PUT, and @CTL
        LD      HL,P0_FLAGS
        RW
        JP      C,P0_GET        ; Service a @GET call
        RW
        JR      Z,P0_PUT        ; Service a @PUT call
P0_CTL  LD      A,C     ; Get control code
        OR      A               ; Zero status request
        JR      Z,P0_CTL_00  ; Jump if so
        DEC     A               ; test for 1
        DEC     A               ; test for 2
        JR      Z,P0_CTL_02; initialise driver
        DEC     A               ; test for 3
        JR      Z,P0_CTL_03  ; clear buffers
        XOR     A               ; Normal return
        RET
P0_CTL_00       ; Test device status
        LD      A,(P0_TXB+RB_B_COUNT)       ; Get
   output buffer byte counter
        RW
        OR      A ; Set flags, ready if no chars
        IN0     A,P0_P_STATUS; return status
   register of ASCI
        RET
P0_CTL_02   ; driver initialisation
        IN0     A,P0_P_STATUS
        AND     255-ST_M_TIE-ST_M_RIE
        OUT0    P0_P_STATUS,A
        IN0     A,P0_P_CA
        AND     255-CA_M_MPE-CA_M_RTS0-
   CA_M_MPBREFR-CA_M_MOD1-CA_M_MOD0
        OR      CA_M_RE+CA_M_TE+CA_M_MOD2
        OUT0    P0_P_CA,A
        IN0     A,P0_P_CB
        AND     255-CB_M_MP-CB_M_CTSPS-
   CB_M_SS2-CB_M_DR
        OR      CB_M_SS1+CB_M_SS0
        OUT0    P0_P_CB,A
;Turn on recv interrupt when DCD goes active
P0_CTL_02_0 IN0     A,P0_P_STATUS
        AND     ST_M_DCD0
        JR      NZ,P0_CTL_02_0
        IN0     A,P0_P_STATUS
        OR      ST_M_RIE
        OUT0    P0_P_STATUS,A
        LD      (HL),0          ; clear flag bits
P0_CTL_03   ; clear buffers
        RES     PAUSE,(HL)
        DI              ; Disable interrupts
        XOR     A
        PUSH    IY
        LD      IY,P0_RXB
        RW
        LD      (IY+RB_B_COUNT),A
        LD      (IY+RB_B_INPUT),A
        LD      (IY+RB_B_OUTPUT),A
        LD      IY,P0_TXB
        RW
        LD      (IY+RB_B_COUNT),A
        LD      (IY+RB_B_INPUT),A
        LD      (IY+RB_B_OUTPUT),A
        POP     IY
        EI
        RET
        RET
P0_PUT          ; Put a character
        LD      IX,P0_TXB       ; pt to xmit CB
        RW
```

```
P0_PUT_LOOP LD     A,(IX+RB_B_COUNT)   ; Get
   count of pending characters
      CP     0FEH   ; Test for upper limit
      JR     NC,P0_PUT_LOOP      ; Loop until
   buffer a less full
      DI
      RB_PUT C
      LD     A,(IX+RB_B_COUNT) ;get char count
      DEC    A      ; Test for 1 byte in buffer
      CALL   Z,P0_ITX      ; Start xmit
      RW             ; interrupt routine
      EI
      XOR    A      ; Set return status
      RET
P0_GET LD    IX,P0_RXB     ; Get a character
      RW
      LD     A,(IX+RB_B_COUNT)   ; count of
   waiting characters
      OR     A             ; Test it
      JR     Z,P0_GET_1    ; Go if none waiting
      DI             ; Disable interrupts
      RB_GET
      LD     A,C
      EI
      CP     A      ; Set zero flag to indicate
      RET    ; Byte received and RETURN
P0_GET_1    OR     1; Return zero with NZ flag
      LD     A,0
      RET    ; To say no character found
P0_INTERRUPT
      IN0    A,P0_P_STATUS; get status register
      RLA
      CALL   C,P0_RDRF     ; receive data ready
      RW
      IN0    A,P0_P_STATUS; get status again
      RRA
      RRA
      CALL   C,P0_TDRE     ; xmit buffer empty
      RW
      IN0    A,P0_P_STATUS; clear possible
   dcd interrupt
      IN0    A,P0_P_CA     ; get ctrl reg A
      OR     CA_M_MPBREFR ; set error flag
   reset bit
      OUT0   P0_P_CA,A     ; send to register
      RET
;Place incoming char into the receive buffer
;(Note chars received when the buffer is full
;will cause a buffer wrap-around and loss
; of one buffer's worth of data)
P0_RDRF
      IN0    A,P0_P_RDR    ; get character
      LD     HL,P0_FLAGS   ; point to flags
      RW
      BIT    PASS,(HL); Test PASTHRU flag
      JR     NZ,P0_RDRF_2 ; If set then
   don't test character
      CP     XOFF   ; Test for XOFF request
      JR     NZ,P0_RDRF_1 ; If not, go ahead
      SET    PAUSE,(HL)    ; Set the pause flag
      RET
P0_RDRF_1 CP XON   ; Test for XON request
      JR     NZ,P0_RDRF_2 ; If not, jump ahead
      RES    PAUSE,(HL)    ; Reset pause flag
      JP     P0_ITX ; Initialise transmission
      RW
P0_RDRF_2 PUSH BC ; save character in buffer
      PUSH   IX
      LD     IX,P0_RXB
      RW
      LD     C,A
      RB_PUT C
      POP    IX
      POP    BC
      RET
;P0_TDRE, routine to handle interrupt by ASCI
;upon character fully transmitted to remote
;device. i.e. transmit buffer is now empty.
P0_TDRE
      LD A,(P0_TXB+RB_B_COUNT) ; Get counter
      RW
      OR     A             ; Test it
      JR     Z,P0_TDRE_1   ; Buffer empty, jump
      LD     A,(P0_FLAGS) ; Get flag bits
      RW
      BIT    PAUSE,A       ; Test pause bit
      JR     NZ,P0_TDRE_1 ; Pause mode on, go
      PUSH   BC     ; get character from buffer
      PUSH   IX
      LD     IX,P0_TXB
      RW
      RB_GET
      LD     A,C
      POP    IX
      POP    BC
      OUT0   P0_P_TDR,A    ; send character
      RET
P0_TDRE_1   ; disable transmit interrupt
      IN0    A,P0_P_STATUS
      AND    255-ST_M_TIE ; turn off
      OUT0   P0_P_STATUS,A ;interrupt enable
      RET
;P0_ITX, Initialise transmission - enables
;interrupt for ASCI transmit register empty,
;causing the interrupt routine to be called.
P0_ITX      ; turn on interrupt enable
      IN0    A,P0_P_STATUS
      OR     ST_M_TIE
      OUT0   P0_P_STATUS,A
      RET
      RW_E
P0_RXS      DEFL   256
      DC     RB_S_BLOCK,0
P0_RXB      DC     P0_RXS,0
P0_TXS      DEFL   256
      DC     RB_S_BLOCK,0
P0_TXB      DC     P0_TXS,0
P0_LAST     DEFL   $-1
P0_LENGTH   DEFL   $-P0_BASE

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;     P1/ASM
P1_DOCALL   IN_WHERE      ; find IN module
      RET    NZ     ; failed to find it
      PUSH   HL     ; save ptr to IN module
      LD     HL,P1_LINK
      CALL   I_LINK ; grab  @icnfg vector
      LD     DE,P1_BASE    ; module start
```

```
      LD    BC,P1_LENGTH ; module length
      LD    IY,P1_RW     ; module relo table
      CALL  I_MOVE ; move module to sys mem
      CALL  I_DCB        ; set up dcb
      POP   DE     ; restore IN module ptr
      CALL  P1_INSTALL   ; install P1 module
      LD    DE,P1_INIT   ; install and
      CALL  I_INIT       ; invoke init
      CALL  I_CONFIRM    ; confirm install
      LD    HL,0
      RET
P1_INSTALL  PUSH  BC
      PUSH  HL
      LD    HL,P1_INTERRUPT     ;int handler
      ADD   HL,BC ; conv to installed addr
      EX    DE,HL
      LD    BC,P1_IV-IN_BASE    ; offset
   within IN module
      ADD   HL,BC ; point to vector
      LD    (HL),E ; store our interrupt
      INC   HL     ;handler address
      LD    (HL),D
      POP   HL
      POP   BC
      RET
      RW_B  P1_RW  ;Driver header and code
P1_BASE     JR    P1_START
      DW    P1_LAST
      RW
      DB    3,'_P1'
      DW    0,0          ; required for DOS
P1_INIT     ;Init status register
      IN0   A,P1_P_STATUS
      AND   255-ST_M_TIE-ST_M_RIE
      OUT0  P1_P_STATUS,A
      IN0   A,P1_P_CA    ;Init control reg a
      AND   255-CA_M_MPE-CA_M_MPBREFR-
   CA_M_MOD1-CA_M_MOD0
      OR    CA_M_RE+CA_M_TE+CA_M_MOD2
      OUT0  P1_P_CA,A
      IN0   A,P1_P_CB    ;Init control reg b
      AND   255-CB_M_MP-CB_M_CTSPS-
   CB_M_SS2-CB_M_DR
      OR    CB_M_SS1+CB_M_SS0
      OUT0  P1_P_CB,A
      IN0   A,P1_P_STATUS       ;Turn on
      OR    ST_M_RIE            ;receiver
      OUT0  P1_P_STATUS,A       ;interrupt
P1_LINK     RET           ;@ICNFG Vector
      DW    0
P1_START    ; Driver code executed by call to
            ; SVC's @GET, @PUT, and @CTL
      JP    C,P1_GET      ; Service @GET call
      RW
      JR    Z,P1_PUT      ; Service @PUT call
      RET
P1_PUT                    ;Put a character
      IN0   A,P1_P_STATUS
      AND   ST_M_TDRE     ;xmit DR empty flag
      JR    Z,P1_PUT      ; until it is set
      OUT0  P1_P_TDR,C    ; send character
      XOR   A             ; worked ok
      RET
P1_GET                    ;Get a character
      LD    IX,P1_RXB
      RW
      LD    A,(IX+RB_B_COUNT)   ; count of
   waiting chars
      OR    A             ; Test it & jump
      JR    Z,P1_GET_1    ; if none waiting
      DI                  ; Disable interrupts
      RB_GET
      LD    A,C
      EI
      CP    A       ; Set Z flag to indicate
      RET   ; Byte received and RETURN
P1_GET_1    OR    1; Return zero with NZ flag
      LD    A,0
      RET   ; To say no character found
P1_INTERRUPT
      IN0   A,P1_P_STATUS; get status reg
      RLA
      CALL  C,P1_RDRF     ; recv data ready
      RW
      IN0   A,P1_P_CA     ; get control reg a
      OR CA_M_MPBREFR ;set err flg reset bit
      OUT0  P1_P_CA,A     ; send to register
      RET
;Place the incoming char into the recv buffer
P1_RDRF IN0 A,P1_P_RDR    ; get character
      PUSH  BC
      PUSH  IX
      LD    IX,P1_RXB
      RW
      LD    C,A
      RB_PUT C
      POP   IX
      POP   BC
      RET
      RW_E
P1_RXS      DEFL  256
      DC    RB_S_BLOCK,0
P1_RXB      DC    P1_RXS,0
P1_LAST           DEFL  $-1
P1_LENGTH   DEFL  $-P1_BASE
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;     IN/ASM
IN_DOLD     HL,IN_LINK
      CALL  I_LINK ; get copy of @icnfg
      LD    DE,IN_BASE    ; module start
      LD    BC,IN_LENGTH ; module length
      LD    IY,IN_RW      ; module relo table
      CALL  I_MOVE_LOW    ; module to sys mem
      CALL  IN_INSTALL    ; set sys vectors
      LD    DE,IN_INIT
      CALL  I_INIT ; install and invoke init
      CALL  I_CONFIRM     ; confirm install
      LD    HL,0
      RET
 IN_INSTALL PUSH  HL     ;Allocate memory
      CALL  I_GET_HILO$  ;for interrupt
      PUSH  HL           ;vector table
      LD    E,(HL)
      INC   HL
      LD    D,(HL)
      EX    DE,HL ; move to appropriate
      LD    A,L   ; 32byte boundary
```

```
      AND    11100000B
      CP     L              ;Go if already at
      LD     L,A            ; 32byte
      JR     Z,IN_INSTALL_0      ; boundary
      LD     DE,32
      ADD    HL,DE
IN_INSTALL_0      EX     DE,HL  ; store
      LD     HL,IN_TABLE    ;table base
      ADD    HL,BC
      LD     (HL),E
      INC    HL
      LD     (HL),D
      LD     HL,IN_DIE      ; build first
      ADD    HL,BC          ; seven empty
      EX     DE,HL          ; entries
      LD     A,7
IN_INSTALL_1      LD     (HL),E
      INC    HL
      LD     (HL),D
      INC    HL
      DEC    A
      JR     NZ,IN_INSTALL_1
      EX     DE,HL  ; make ASC0 entry
      LD     HL,IN_ASC0
      ADD    HL,BC
      EX     DE,HL
      LD     (HL),E
      INC    HL
      LD     (HL),D
      INC    HL
      EX     DE,HL  ; make ASC1 entry
      LD     HL,IN_ASC1
      ADD    HL,BC
      EX     DE,HL
      LD     (HL),E
      INC    HL
      LD     (HL),D
      INC    HL
      EX     DE,HL  ; update HILO$
      POP    HL
      LD     (HL),E
      INC    HL
      LD     (HL),D
      POP    HL
      RET
IN_WHERE          ;Where is this module?
      LD     DE,IN_MODULE
      SVC    @GTMOD
      RET    Z
      CALL   IN_DO
      LD     DE,IN_MODULE
      SVC    @GTMOD
      RET
IN_MODULE   DB     '_IN',3
OS$B_LBANK  EQU    0202H
OS$B_OPREG  EQU    0078H
IN_P_IL     EQU    0033H  ; int vector low
      RW_B   IN_RW  ;Module header and code
IN_BASE     JR     IN_INIT
      DW     IN_LAST
      RW
      DB     3,'_IN'
IN_INIT     ;Module initialisation
      LD     DE,0           ; point to table
IN_TABLE    DEFL   $-2
      LD     A,D    ; set cpu int vector
      LD     I,A
      OUT0   IN_P_IL,E
IN_LINK     RET    ;@ICNFG Vector
      DW     0
;     IN_ASCx - Handle ASCI generated ints
IN_ASC0     PUSH   DE
      LD     DE,0
P0_IVDEFL   $-2    ; p0 interrupt vector
      JR     IN_HANDLE
IN_ASC1     PUSH   DE
      LD     DE,0
P1_IVDEFL   $-2
;IN_HANDLE   Handle interrupt by making envi-
; ronment sound before jumping to the routine
; required, since it may exist in paged-out
; memory. (this code derived from code found
; within a patched version of ls-dos in my
; possession; can't remember what patches had
;been applied; probably houde')
IN_HANDLE   PUSH   HL
      PUSH   AF
      LD     HL,OS$B_LBANK;
      LD     A,(HL)         ;
      LD     (HL),0         ;
      LD     H,A            ;
      IN0    L,CBR  ; get current CA1 offset
      PUSH   HL             ; save for later
      XOR    A              ; zero offset
      OUT0   CBR,A
      LD     HL,OS$B_OPREG; image of opreg$
      LD     A,(HL)
      PUSH   AF
      AND    8CH
      OR     03H
      LD     (HL),A
      OUT    (84H),A
      LD     HL,IN_DONE     ; return address
      RW
      PUSH   HL
      EX     DE,HL
      JP     (HL)    ; exec interrupt routine
IN_DONE     POP    AF
      LD     (OS$B_OPREG),A
      OUT    (84H),A
      POP    HL ;rcvr OS$B_LBANK and CBR values
      OUT0   CBR,L  ; restore offset
      LD     A,H
      LD     (OS$B_LBANK),A
      POP    AF
      POP    HL
      POP    DE
      EI
      RET
      RW_E
IN_LAST           DEFL   $-1
IN_LENGTH   DEFL   $-IN_BASE
```

SETCOM to change driver parameters; the parameters are fixed at 4800 baud, 8-bit word length, 2 stop bits, no parity, and the driver does not use the WAKEUP service call interface. Because of this last provision, **the driver will not work with the COMM program**. The second driver which is part of CL3, contains the code necessary to provide a system interface to the ASCI port 0. This driver contains two 256-byte character buffers, but again the driver has no WAKEUP interface, so it too cannot be used for COMM. The third driver used to access ASCI port 1 contains only a receive buffer, and no WAKEUP interface. Both ASCI drivers are hard coded for the same parameters as previously noted.

The implementation James designed, however, permits the ASCI drivers to be installed in memory independently of the SIO driver. This means that you could install the regular COM driver to use with COMM, and then install either the P0 or P1 driver to add additional special-purpose serial ports.

Using any of the trio of drivers provided in CL3 requires you to make modifications to the code if you need parameters other than the hard-coded ones. These parameter changes can be made to the assembled driver program through a series of patches. The SIO driver has two configuration bytes: one at X'3646' alters the baud rate, while another at X'3647' alters the other parameters. Let's first look at the baud rate.

The SIO has a baud rate register (BRG) accessed through port 0E9H. The driver stores the configuration value for this register at X'3636'. There are 16 distinct values which can be set for the receive side and 16 for the transmit side. Although the COM driver provided with the DOS doesn't support different baud rates for send and receive, this is only a software restriction - the hardware is perfectly capable of using a different rate for transmitting and receiving. The BRG register uses the high four bits to control the transmit rate and the low four bits control the receive rate. The next table reflects these rate assignments.

Value (hex)	Rate (baud)
00H	50
01H	75
02H	110
03H	134.5
04H	150
05H	300
06H	600
07H	1200
08H	1800
09H	2000
0AH	2400
0BH	3600
0CH	4800
0DH	7200
0EH	9600
0FH	19,200

Note that the CL3 driver has this configuration byte initialized to 0CCH setting up the transmit and receive side for 4800 baud. If, for instance, you wanted to use 2400 baud, change the value to 0AAH.

The configuration byte at X'3647' contains initializing values for word length, parity, and number of stop bits. The following table depicts these assignments.

Bits	Function	Value
7	Parity	1=EVEN; 0=ODD
6,5	Word Size	01B=5; 10B=6 01B=7; 11B=8
4	Stop Bits	0=1 Stop Bit 1=2 Stop Bits
3	Parity	0=Enable 1=Disable
2	Break	0=Const SPACE 1=normal
1	DTR	0=ON, 1=OFF
0	RTS	0=ON, 1=OFF

Here's where a knowledge of binary comes in handy. Word length, etc., is selected by constructing a byte containing data in each of the bit fields. You need to *build* a value after determining the individual selections. The hard-coded value is 0FFH, or 11111111B, which has all bits enabled. Referencing the above table, you can see this sets EVEN parity (which is not used since parity is disabled), 8-bit word length, 2 stop bits, parity disabled, normal operation, DTR=OFF, and RTS=OFF (DTR and RTS use inverted logic; perhaps James had wanted to set those to ON). If you want to select 7-bit word, 1 stop bit, even parity, DTR=ON, build 10100101B or 0A5H. This is the default setting of the COM driver.

Now let's turn to the XLR80 and its 64180 ASCI ports. The baud rate and parameter setup of these ports are more complex than the SIO port. Examining the baud rate selection first, the actual transmission rate is determined by five bits of a register, the ASCI Control Register B0 and B1, as well as the clock rate used to control the 64180. The XLR8er uses a 12.244Mhz crystal which establishes a 6.144Mhz clock rate. The five Control Register bits are: a prescaler (PS) which is always set to 0 when using a 6.144Mhz clock; a Divide Ratio (DR), and three source/speed select (SS) bits. The table on the following page describes the bit assignments of the Control Register.

Bits 7 and 6 deal with multiprocessor mode; they would be set to 0 for XLR8er installations. Note that bit 5 is used for dual purposes; it is Clear to Send (CTS) on read, and prescale on write. Bit 4 is used for selecting parity even or odd. For these assignments, the baud rate is determined using the following table:

PS	DR	SS2	SS1	SS0	Rate
0	0	0	0	0	38400
0	0	0	0	1	19200
0	0	0	1	0	9600
0	0	0	1	1	4800
0	0	1	0	0	2400
0	0	1	0	1	1200
0	0	1	1	0	600
0	0	0	0	0	9600
0	1	0	0	1	4800
0	1	0	1	0	2400
0	1	0	1	1	1200
0	1	1	0	0	600
0	1	1	0	1	300
0	1	1	1	0	150

Note that some baud rates may be selected

Control Register Table

7	6	5	4	3	2	1	0
MPBT	MP	CTS/PS	PEO	DR	SS2	SS1	SS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

using more than one arrangement of control bits. Since the parity is also selected using this control register, let's look at what is used to select the non-baud rate operating parameters before addressing the methods of changing the driver via a patch.

ASCI Control Registers A0 and A1 contain three bits used to select word length, parity enable, and number of stop bits. These are bits 2-0 and are denoted as MOD2, MOD1, and MOD0 respectively. There configurations is as follows:

MOD2	0=7-bit
	1=8-bit
MOD1	0=Parity disabled
	1=Parity enabled
MOD0	0=1 stop bit
	1=2 stop bits

With this information at hand, let's look at James' source code. For the port 0 driver (P0), control register A0 is initialized with the code at 3ACEH to 3AD7H; control register B0 initialization is from 3AD8H-3AE1H. Both these code fragments utilize the same form:

1. Read a byte from the port
2. Strip selected bits using AND
3. Merge desired bits using OR
4. Write the revised byte to the port.

When you AND byte A with byte B, you essentially strip from byte A, any bit which is a zero value in byte B. James' method of coding constructs the byte-B mask by subtracting from 255 (all one's), the bit-value corresponding to a particular bit assignment. For instance, the mask value for control register A is constructed by subtracting from 255 (11111111B), CA_M_PE (80H leaving 01111111B), CA_M_RTS0 (10H leaving 01101111B), CA_M_MPBREFR (08H leaving 01100111B), CA_M_MOD1 (02H leaving 01100101B), and CA_M_MOD0 (01H leaving 01100100B). The resulting mask value is 64H. Theoretically, to be complete, since you really want to strip all bits which are going to be subsequently either set or reset , this mask should also strip CA_M_MOD2 (04H) setting the mask value to 60H. This would strip all but bit 6 and bit 5 from whatever value was read from the port.

The next instruction at 3AD3H, OR's the result left in the accumulator with the bits desired to establish the operating parameters settable via this register. Hard coded is the ORing with CA_M_RE (40H always needed to enable the receiver), CA_M_TE (20H always needed to enable the transmitter), and CA_M_MOD2 (04H). This establishes MOD2=1, MOD1=0, and MOD0=0 for 8-bit word length, 1 stop bit, parity disabled). If you wanted to select 7-bit word length, 1 stop bit, and even parity, the OR'd value would need to be 62H (MOD2=0, MOD1=1, MOD0=0).

The code from 3ACEH to 3AD7H established the baud rate. The value sent to the control register is, like before, obtained by reading the current value, stripping off some bits, then ORing in the desired ones. To be complete, we should strip all of the bits associated with baud rate selection, then OR in the ones we need to set to establish the desired rate. Jame's code takes a little shortcut since he was specifically using 4800 baud. In general, since we are dealing strictly with the XLR8er, we know that the MP bit must be set to 0, MPBT is irrelevant with MP=0, the PEO bit is used to select parity type, and the five remaining are used to select the baud rate. Thus, we really do not need to read any value but can just output the desired setting. However, to stay with the code and develop a short patch to deal with the operating parameters, I would recommend masking the value READ with 80H (keeping what ever was the value of MPBT), then ORing with a value developed from the baud rate selection table above, and the desired selection of parity. For instance, if you want to select 1200 baud with EVEN parity, you first must have enabled parity at Control Register A0 above, then select 1200 baud here with PS=0, DR=0, SS2=1, SS1=0, SS0=1, with PEO=0 for EVEN parity. The resulting OR value is 00000101B or 05H.

The driver for ASCI port 1 has a similar block of code to establish the parameters for that port. The corresponding code range is 3E7CH to 3E8FH. Lastly as far as parameter patching goes, the corresponding D-patch locations for each of the byte values is identified in the following table.

Byte	**Port-0**	**Port1**
Control A AND	D08,64	D0C,B0
Control A OR	D08,66	D0C,B2
Control B AND	D08,6E	D0C,BA
Control B OR	D08,70	D0C,BC

With these software adjustments out of the way, let's turn to some of the hardware interfacing requirements. To begin with, if you have the XLR8er manual published by MISOSYS (Revision 2.0 dated 04/30/88), make note that there is an error on page 41. Due to some erroneous information provided by Hi-Tech - the original manufacturer of the XLR8er board, the pin assignments for RS2 (Port 2 RS232 DB25 pinout) are inverted. The RX1 assignment should be on pin 3 and the TX1 pin assignment should be on pin 2. Another point is that Hi-Tech stated that the Request to Send (RTS) lead should be tied to DTR. This is not correct. To begin with, RTS is an output of the RS232 towards the host. Tying RTS to DTR would make sense if the host was incapable of setting DTR and the RS232 needed to see DTR set TRUE. By setting RTS TRUE, a connection from RTS to DTR would make DTR appear TRUE. Unfortunately what is needed is to have CTS0 set TRUE. According to the 64180 User's Manual, "if the CTS/ input pin is HIGH, the TDRE bit is inhibited (i.e. held at 0)." It states

further about TDRE (transmit data register empty), that "TDRE is cleared to 0 until the ASCI transfers the byte from the transmit data register to the transmit send register". In other words, as long as there is no clear-to-send condition, the ASCI will be unable to send.

James Cameron discusses the 64180's use of data carrier detect (DCD) briefly in Note 2. Do not gloss over this note as it is extremely important. Port 0 is unable to operate without DCD acknowledged to be TRUE, as the 64180 will continuously interrupt until that condition is met. That's why he coded the port 0 driver initializer (see address range 3AE2H to 3AE8H) to loop until DCD is TRUE, before enabling receiver interrupts. In typical operation of a connected serial device, you must ensure through some means that the DCD line is held TRUE.

The necessity for requiring CTS and DCD to be TRUE for operation of ASCI port 0 requires you to make some choices. If you are connecting to a modem, that usually provides a CTS signal. For instance, the Hayes Smartmodem does; and the TT512P internal 4P modem sold by MISOSYS does. The Hayes Smartmodem also provides a switch option to force DCD TRUE; the TT512P does not! However, if you are going to use port 0 to connect to another computer through a null modem cable, you must provide some way to enable both the CTS and DCD leads. Because of this, **it would make sense to tie the DTR lead to both CTS and DCD**.

My testing of the driver utilized two Model 4 computers. I modified the cable sets, which I had originally purchased from Hi-Tech, by connecting DTR (pin 20) to both DCD (pin 8) and CTS (pin 5). I then added the two serial converter chips (the 1488 and 1489) to the XLR8er board. Lastly, I connected 12V power to the serial converter chips by adding a jumper power cable to XLR8er port P3 tapping the power from the RS232 SIO power cable on the Model 4. Again with this cable, I had to correct a wiring error in the original set of power cables Hi-Tech had had manufactured, and which I had purchased. Once the XLR8er changes were accomplished, I routed the revised serial interface cable through the computer case bottom and connected it to the XLR8er board. My 4P had a similar adaptation with the serial interface cable routed out through the modem cover plate. I then used a null modem cable to connect the two computers.

For the computer with the CL3 driver, I installed both the P0 and P1 drivers with these commands:

```
SET *P0 CL3 (P0)
SET *P1 CL3 (P1)
```

I then set up that computer as a simplified host by issuing the commands:

```
LINK *KI *P0
LINK *DO *P0
```

On the computer at the other end of the null modem cable, I installed the COM driver; used SETCOM to set it up for 4800 baud, 8-bit word length, no parity, and 1 stop bit to conform with the hard coded parameters of the CL3 driver; then invoked the COMM communications program. I was then able to control the first computer from COMM. If I wanted to get fancier, I could have installed the HOST portion of LS-HOST/TERM on the XLR8er equipped computer, then used the TERM portion of LS-HOST/TERM on the second computer. This would have enabled me to fully support direct video VDCTL service calls through the serial interface. However, the initial testing did not require this sophisticated arrangement.

Based on the driver supplied by James Cameron, the XLR8er user should be able to utilize it directly for certain purposes, taking the idiosyncrasies of the hardware interface into mind, and patching the operating parameters according to your own needs. The experienced assembly language programmer should be able to utilize CL3 as a useful base to easily adapt the driver to serve to your own needs. If you do not need buffering, you could add the WAKEUP service call interfacing (using *THE SOURCE* as a guide) and remove the buffering and buffering overhead. James Cameron has accomplished the hard work - that of understanding the 64180's Asynchronous Serial Communications Interface (ASCI) and writing the code necessary to use both its transmit data register empty (TDRE) and receive data register full (RDRF) interrupts.

For hardware support, MISOSYS is making available the following kit of parts at a price of $40; we have only 16 kits available:

- serial interface cable, approximately 20" long, consisting of a 20-pin header connector on one end (to connect to XLR8er header P2, and split into two DB25 female connectors on the other end bringing out serial ports RS1 and RS2 (i.e. 64180 ASCI ports 0 and 1). RS2 presents TX1, RX1, and GND on pins 2, 3, and 7 respectively; RS1 presents TX0, RX0, and GND on pins 2, 3, and 7 respectively, with RTS0 on pin 4, CTS0 on pin 5, DCD0 on pin 8, and DTR on pin 20.

- 12-volt power interface cable, approximately 15" long, consisting of a 4-pin AMP female connector on one end (plugs into XLR8er P3 connector), and both a male and female 4-pin AMP connector on the other end (used to tap into the Model 4 motherboard +/-12V.

- both a 1488 and a 1489 RS232 interface chip set.

PRO-WAM

What's It All About

by Roy Soltoff

Over the years, much has appeared in print concerning our PRO-WAM product. However, I was recently asked to provide an explanation of PRO-WAM in an upcoming issue of *The MISOSYS Quarterly* so as to enable more recent users an opportunity to understand just exactly what PRO-WAM can do for them. To that end, this article addresses the evolution of PRO-WAM from its infancy to its maturity at this tender age of seven years of longevity.

A Brief History of Computers

The first electronic computer is generally considered to be the ENIAC, which was completed in 1946 at the University of Pennsylvania Moore School of Engineering. The project was under the direction of two physicists, J. Presper Eckert and John William Mauchly. To put this computer into perspective, it consisted of more than 18,000 vacuum tubes and 1500 relays. The computer was *hard-wired* to perform the task to which it was designed, that of computing ballistic trajectories. ENIAC was capable of performing nearly 5000 additions per second. The fact that the machine had to be re-wired for each change in program, required a great deal of set-up time. Nevertheless, it was in use for about ten years.

The next stage of development came when Hermine H. Goldstine and John Von Neumann joined the team to develop the EDVAC computer - the first to utilize a program stored in memory. That project was almost doomed when Eckart and Mauchly left the Moore School to start their own company, the Eckart-Mauchly Corporation, to design and manufacture the UNIVAC (UNIVersal Automatic Computer) computer. In 1950, Remington Rand acquired the company; it's now known as UNISYS. UNIVAC I was first delivered in 1951 and used by the Bureau of the Census for its 1950 results. I remember visiting the Remington Rand plant in the late-50's when the Univac II was being built. Von Neumann is the one generally credited with the inclusion of a program stored in memory.

Von Neumann, Goldstine, and other collaborators at the Institute for Advanced Studies at Princeton began developing a new computer model referred to now as the *IAS* or *Von Neumann* machine. Most modern day computers are still based on this design. The Von Neumann architecture is classic. Because of this point, it is wise to discuss a little bit about the architecture.

The Von Neumann machine typically has five components: an input unit, a memory unit, an arithmetic-logic unit, a control unit, and an output unit. The control fetches an instruction, decodes it, and takes the appropriate action. It is important to note that only one instruction is executed at a time; the entire process is conceptually a *serial* one.

In todays microcomputers, their computation processing is still a Von Neumann architecture. This means that only one action is processed at a time. The computer is engaged in only one task at a time. As computers got faster and faster in their processing, it became a challenge to design the input and output units to be able to keep up with the processor. In fact, in most cases, the processor is just sitting around waiting for a slow human or a slow peripheral to do something. Most tasks fall into this kind of activity in which the amount of processing needed is small relative to the input and output; the task is *I/O bound.*

There are other types of tasks which are *compute bound*, i.e. they demand a great deal of processing with relatively little output. Typically the applications are scientific in nature, such as weather data assimilation and processing performed to produce probability forecasts, and real-time simulation of complex processes. For these types of tasks, supercomputers have been developed which generally use multiple processors and an architecture called *parallel processing*. As a simple example, consider the calculation of the mathematics problem, 3 x 7 + 4 x 8. In a Von Neumann machine, 3 is multiplied by 7 then the result is temporarily stored, then 4 is multiplied by 8 with that result in term added together with the previously stored result of the first multiplication. Every math operation is performed sequentially. In a parallel processor, the problem can be broken up into sub-problems. Calculation of the first product is carried out on one processor simultaneously with the calculation of the second product on another processor. A complex control program is essential to properly break up the master problem into separate problems which can be processed simultaneously.

But parallel processor machines are complex, expensive, and most tasks are not capable of being partitioned; thus, most machines are of the Von Neumann architecture and incur a great deal of idle time. In order to be able to utilize the processor during this idle time, operating system developers designed a means to partition the computer resources into areas such that each area could contain one task, and have a control facility which would automatically switch the processor to each different task in turn. In the L(S)-DOS environment, there is a task processor which controls the automatic switching

between one *major* task and multiple *minor* tasks. A major task may be the word-processing program you are using to write that letter to Aunt Mary. Or it could be the data base you are using to print out that list of address labels. Most users are not aware of the minor tasks going on; however, if you are using the system spooler, the activity of despooling your printed material is actually a small task being controlled by the task processor. If you are displaying the clock time on your video screen, it is being displayed by a small task under control of the task processor. Keyboard type-ahead is another small task. The actual time-keeping of the real-time-clock is managed by a small task controlled by the task processor.

But suppose you wanted to perform two major tasks? Obviously, on a Von Neumann machine, you could not process them simultaneously! However, if there were a means to partition the computer's resources into multiple areas of size each sufficient to hold a major task, then you could load up the computer with a word-processor in one area, a data base program in another, a spreadsheet in a third, etc. If the computer had a means of automatically switching between each task, the computer is then referred to as having a *mult-tasking* capability. The time to switch between tasks must be quite small relatively to the processing needs of the tasks. Typically, if the partitioning facility is available but the switching between partitions is performed manually, the system is referred to as having a *task-switcher*.

Task switching conveys a considerable benefit to the computing user. People's lives are usually not sequential in the carrying out of a task. Whether we are at a business setting or the home environment, a great deal of interruptions occur while we are doing something. We may be typing a letter with a word processor when a telephone call comes in requiring us to consult our calendar agenda. If the calendar agenda is a program on our computer, we may just have to save the letter to disk, exit the word processor, load up the calendar program, then exit it to reload our word processor then our letter - as well as return to the exact spot we were typing into. That takes time. Folks generally do not like interruptions. So a task switcher permits you to temporarily switch away from what you are doing to use your computer for something else. When the interruption is satisfied, you can easily switch back to what you were doing.

On the TRS-80 Model 4, at least two forms of task switchers are currently available. One is DoubleDuty. This task switcher, available form MISOSYS, divides a 128K machine into three partitions - two of which are full-sized 64K operating environments capable of running a complete program such as a spreadsheet, data base, or word-processor. The third partition allows you to invoke DOS library commands. Switching between the tasks is accomplished by simply pressing a particular key combination. In this case, if you are writing a letter in one partition, you can switch to the second to bring up a data abase record, or whatever else you need your computer for. Another task switcher for the Model 4 is PRO-WAM, also available from MISOSYS. This task switcher uses one 32K RAM bank of a 128K machine and provides the capability of manually interrupting a major task to allow operation of a multitude of medium sized specialized applications provided as part of the PRO-WAM package. In addition, PRO-WAM also includes the ability to cut and paste text between the interrupted program and any of its applications. PRO-WAM was designed to be extensible, which means that a relatively straightforward process exists for the assembly-language programmer to write his/her own applications.

This brief history of computers now provides the background necessary to approach the next topic.

ROOTS

International Business Machines (IBM) introduced its *Personal Computer* in August of 1981, with a DOS provided by Microsoft Corporation. Just like the Model 4 LS-DOS originally provided by Logical Systems was called *TRSDOS* by Tandy, the Microsoft MS-DOS was called *PC-DOS* by IBM. The three magical letters of I-B-M obviously aided in making the PC the success story of the 80's. It also didn't hurt that IBM provided the machine with an *open-systems* architecture, complete with all hardware specifications and printed source code of the Basic Input Output System (BIOS). After competing companies stumbled when they introduced similar but not totally compatible products, the PC finally took off with sales through the roof when the industry shifted to manufacturing machines which were virtually identical in operation to the now standard IBM-PC. These identical machines were termed *clones*. The microcomputer had reached critical mass for success

One thing necessary to point out with the PC is that, like the TRS-80, it was and still is a Von Neumann machine. The PC processed only one major task at a time. About the only secondary task provided with the DOS system was a spooler, and its means of operation were not part of the publicly disclosed documentation. It took a small upstart company by the name of Borland to decipher the means which enabled the operation of the spooler and turn it into a product sensation that created a brand new product category. Borland's original claim to fame was an exceedingly popular PASCAL compiler known as *Turbo PASCAL*. Borland had a core of gifted programmers. Now programmers by nature are quite curious; they like to solve puzzles. One of those puzzles was, of course, PC-DOS, (now generally referred to as MS-DOS) the operating system on the PC. Borland was the first to decode the workings of the spooler and to capitalize on that discovery with a commercial product in 1984. What they discovered were the tricks necessary to create a *terminate and stay resident* (TSR) program. The TSR is no different from the installable device drivers and filters available under L(S)DOS since its inception - TSR is just a fancy new name. However,

with the discovery of how to write a TSR program, and with the availability of a much larger addressable address space in the PC compared to the older 8-bit computers, Borland was able to write a major program called **SIDEKICK™** and allow it to be resident along with the main program running on the PC. Borland assembled all of the usual desktop paraphernalia (i.e. calendar, a calculator notepad, a programmers conversion chart, a telephone list, etc.) into a single cohesive program, made it able to reside totally in memory along with any other main program, and enabled it to be activated at the touch of a button to be popped up onto the screen. Along with this program was the means to switch between the main program and the memory-resident SIDEKICK program. Now besides word-processing, data base, and spreadsheet, you had memory-resident desk-top pop-ups.

SIDEKICK was able to capture data from the video screen and pass the data to one of its applications, as well as pass data from an application back to the main program. This cut and paste between programs was termed import and export depending on the direction of the data movement. The intrigue of this new class of program, and the resulting publication of the newly discovered methods of writing TSR's, spawned many others to develop similar competing programs. Perhaps the ton of money Borland was raking in with sales of SIDEKICK didn't hurt either. The PC world was soon awash in SIDEKICK look-alikes, and work-alikes.

ENTER PRO-NTO

Karl A. Hessinger was one of my freelance programmers who became intrigued at the SIDEKICK program. He began to develop a SIDEKICK-like program for the TRS-80 and came to me with his preliminary implementation. His *SK* was implemented using slightly less than 6K of high memory; included a calendar, a card filer, an auto-dialer, and an RPN calculator; provided cut and paste, and popped up its applications in small windows on the screen. The applications and saved screen images were stored in one available 32K RAM bank which required a 128K machine. Together we revised the SK architecture to use the library overlay region for each application's execution. This allowed the implementation to use but 3K of high memory and made it extensible by allowing for easily written external applications which could be invoked. In addition, I worked with Karl to generate additional applications to be included with the package. These included a character set display, an algebraic calculator, and address mail list, a bring up file, a small terminal program, and an external data file sort utility.

During this time period, I was using a convention of naming all Model 4 MISOSYS programs with a prefix of "PRO", which essentially stood for *professional* products. We obviously couldn't use SK as the name of this product, so I came up with the name, "PRO-NTO". In December of 1984, I pre-announced the package with this coverage in *NOTES FROM MISOSYS*, Issue IV; the announcement notes the derivation of the name:

"MISOSYS is going to do something it rarely does - pre-announce a product. PRONTO is not expected to be released until March 1985; however, the internal development prototype of this package is so outstandingly significant, that I felt it most important to let you know what is on the immediate horizon. PRONTO sounds like TONTO - whom we all know was the sidekick of The Lone Ranger. Does that perk you up. PRONTO conjures up visions of being fast - i.e. pop up here, pronto! Pop up you say?"

"PRONTO is for the Model 4/4P 128K machine. Pronto provides a WINDOW SuperVisor Call (SVC) to permit the use of overlayed windows from any and all applications. Pronto is a window application manager which supports function key keystroke invocation of up to FIVE small applications plus a LIBrary EXEC facility which gives direct access to all of the DOS's library commands. PRONTO requires one 32K RAM bank, about 2K of high memory, and a small piece of low RAM. We feel that PRONTO is going to be so useful and important to EVERY 128K Model 4/4P user and every systems house writing applications for the 4/4P that we are going to provide preliminary specifications for PRONTO. In addition, due to the expected popularity of PRONTO, you may wish to place a firm pre-paid order for PRONTO now. All orders for PRONTO will absolutely be filled in the sequence that they are received. Here's its capabilities."

"Pronto allows an application to request a window on the current screen of whatever size you need - up to the maximum of 80 by 24. Windows requested of size 78 by 22 or smaller will have a box automatically drawn by the window manager. Under the window application environment, up to 4 windows may be nested, i.e. an application can invoke more than one window or nest to another application which requests a window. More windows may be available when operating solely under the window mode."

"The window control SVC (@WNCTL) provides nine functions of which the first six are similar to @VDCTL: WNPEEK - character window peek, WNPOKE - character window poke, WNSETCUR - set cursor position in window, WNGETCUR - obtain cursor position in window, WN2WIN - mover buffer image to window, WN2BUF - move window image to buffer, WNCREATE - request a window of size and position, WNCLOSE - close current window and revert to previous window, WNDSP - display character at window's cursor, and WNDSPLY - display string at window's cursor. The window display driver supports CR, LF, Erase to EOWL, Erase to EOW, and destructive backspace."

"The PRONTO application manager provides support of application programs which run totally within the DOS's library overlay region and do not exceed 2K in length. Five applications may be resident at one time and are stored in the 32K RAM

segment. Applications run in the address range 2800H-2FFFH and have available two 512K data blocks from 2600H-27FFH and 2400H-25FFH. Where the program requires less than the 2K of code space, it has more space available for data. Pronto saves this 3K address space on each application invocation and the contents of the video screen are saved when a window is requested. Applications can be invoked recursively or consecutively. The application manager even displays a window menu describing each application available at a keystroke. The applications are loaded from core-image files during the installation of PRONTO."

"PRONTO comes with its own set of applications. A perpetual calendar application can display a month at a glance - for any year since the adoption of the Julian calendar and for as many years as you care to project (there is some upper limit; however, its thousands of years away). PRONTO comes with a programmer's reverse polish notation calculator and a standard algebraic one for you normal folks out there. PRONTO comes with a notepad capable of keeping all of those scratchpad entries in order. Pronto comes with an Autodialer usable with a Hayes-compatible modem. This application doubles as an address file. The DOS library command executive is included as one of the six built-in applications. Other applications are planned for release."

PRO-NTO was released on schedule. The first in-depth review appeared in the November 1985 issue of *80 Micro* magazine. The review, entitled PRO-NTO: TRSDOS 6.X's Sidekick, is reprinted in almost its entirety here. It pertained to version 1.0, thus material not pertinent to the most recent release 2.x release of PRO-WAM has been deleted. Nevertheless, the review conveys a good understanding of what PRO-NTO, and now PRO-WAM, has to offer. Here's what one user, Gregory Cleland, had to say about Harrell's review of PRO-NTO. He wrote, "*I've had my new copy of PRO-NTO for a couple of days now and I've got to say that this is the best price/performance package that I've ever bought. As usual for MISOSYS products, the documentation was excellent and the software stunning. John Harrell's review in 80 Micro is right on target. PRO-NTO is a five star product.*

PRO-NTO was our most successful non-DOS product, although not anywhere successful as I thought it should have been. Perhaps there were a great deal of copies in use which were never purchased??? Nevertheless, PRO-NTO became noticeable to Chemical Bank. In February of 1986, MISOSYS received the following letter from the legal firm of Davis Hoxie Faithfull & Hapgood:

Dear Sir:

We are trademark counsel for Chemical Bank and Chemical New York Corporation, and their wholly-owned subsidiaries, Chemical Technologies Corporation and Pronto U.S.A., Inc., owners of the PRONTO electronic banking and information service and the PRONTO trademark and service mark. Chemical New York Corporation is the owner of a number of United States trademark registrations for this mark, including one for computer programs. Copies of these registrations are enclosed, along with a copy of the label on the PRONTO software disk which Chemical distributes to its PRONTO customers.

The PRONTO electronic banking and information service enables a bank customer using the PRONTO floppy disk and a personal computer, linked by a modem to the bank's data processing center, to review the status of his or her accounts, transfer money between accounts, pay bills and send messages to the bank and other PRONTO users. A copy of the current PRONTO brochure describing this service is enclosed.

Chemical Bank is a pioneer in the development of electronic banking, and its PRONTO system and PRONTO service mark and trademark are widely known. The PRONTO system and marks are licensed to other banks nationwide.

Your company and its PRONTO software recently came to our clients' attention. We believe that your use of PRONTO in connection with the sale of this product will likely cause confusion with our clients' PRONTO products and services. We ask, therefore, that your company not use PRONTO in connection with the sale of software, or in connection with the sale of any similar product or service. Please reply promptly with your assurance that you are taking prompt steps to cease use of the PRONTO mark.

With the information supplied to us, Chemical Bank received their first trademark on "PRONTO" on May 31st, 1983 "for computer programs recorded in a memory cartridge for use in personal and small business banking and financial programs". Chemical's *Pronto* was an electronic banking system, that interestingly enough, I believe is now defunct. Our attorney investigated the purported claims and advised us as to the position we should take. Even though our use of name for our product was in form "PRO-NTO", we nevertheless felt it imprudent to wage battle with Chemical Bank and looked for another name. We then addressed the following to Chemical Bank's attorney.

"After reviewing the documents supplied by you and the documents associated with the announcement of our PRO-NTO product in December of 1984, our corporate attorney has advised us that we "should change the designation for our software." Counsel further added that, 'Some of the ads [MISOSYS] provided ... clearly would be confusing when compared to the use of those words by the Chemical Bank.' Thus, I have been advised by counsel to write you indicating that we will take the necessary steps to cease using the PRO-NTO (or PRONTO) designation for our software.

In light of these findings, I have instructed our attorney to perform a trademark search for a new name that has been selected for our product. As soon as we can determine that the selected name does not infringe on any other company, we will proceed to

alter our advertising and other product information concerning our windowing and application manager product.

Since advertising leadtimes are in excess of two months, and a trademark search may take some time to accomplish, I have instructed our advertising department to incorporate the following notation in our PRO-NTO ad as soon as possible. "Coming soon: a new name for PRO-NTO - to avoid confusion with Chemical Bank's Pronto electronic banking system".

We apologize for any inconvenience our use of Chemical's trademark may have caused. However, at the time we selected the PRO-NTO name for our product in 1984, Chemical's electronic banking system was certainly not an item known by us to be a computer software program."

I explored various names for the PRO-NTO product such as "PROTON", a variation on the letters "PRO-NTO" and an implication of being at the *core* of your machine. But there turned out to be a PROTON corporation, and that name was discarded. Other names considered were: PRO-DIGY (meaning a person with exceptional talents), PRO-FOUND (I think you know what that means), PRO-MISE (a declaration of assurance), and WAM! We then enlisted the services of a patent attorney to investigate other names which we could use for our "PRO-NTO" product. We selected "WAM!", "PRO-DIGY", and "PRO-MISE". It turns out that all of those were no good. The attorney turned up names such as , "THE PROMISED LAN", PROMIS and PROMISE for computer software, PRODIGY for electronic musical synthesizers and PRODIGY for microcomputers; some of the PRODIGY trademarks were owned by Prodigy Systems Inc. WAM turned up for a laundry detergent and office printing machines, with various other derived uses of WAMS such as WAMS/LOCATOR "for business and legal conflict of interest computer program instruction manuals", WAMS/SAFE for "computer programs for file management", not to mention "WHAM!" for entertainment! This, of course, put a serious damper on my enthusiasm. We wound up spending about $500 to exhaust all of our reasonable choices. After the trademark search, I finally decided that "PRO-WAM" was sufficiently different from all the other *WAMs*, and proceeded to put it to use. The "WAM" part comes from "Window and Application Manager". The PRO indicates a TRSDOS 6 product.

In the Summer of 1986, the first issue of *The MISOSYS Quarterly* announced the name change from "PRO-NTO" to "PRO-WAM" with this notice: *"You will soon see this term used in our advertising. Don't worry, it's still the same dependable PRO-NTO product (not to be confused with electronic banking)."*

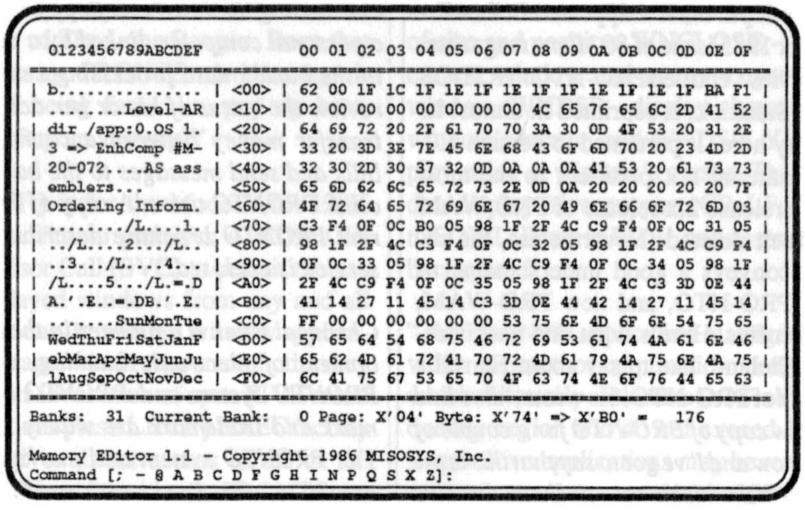
Of course you must understand that it is quite difficult to re-name a product. I remember when Nissan Motors of Japan changed the name used in the United States from "Datsun", as in "Datsun 280Z", to Nissan, as in "Nissan 300 SX" - or is it 300 XYZ? In any event, the process was spread over a number of years with both names being used simultaneously (as in Datsun to Datsun/Nissan to Nissan/Datsun to Nissan). That's sort of similar to changing FOOD to WART in four easy steps, each step changing but one letter: FOOD, FORD, FORT, FART (excuse me), WART.

ENTER MISTER ED

Meanwhile, we were busy at work building a separate package of editor applications which was released in the Fall of 1986. This product was called Mister ED, and included a bundle of editors for different tasks. Mister ED was loaded with seven new applications for use with our Window and Applications' Manager, PRO-WAM. You would get a file editor (FED), a disk editor (DED), a memory editor (MED), a video screen editor (VED), and a text editor (TED). All were full screen editors which made your editing jobs easy. Best of all, these were all PRO-WAM applications so they were usable whenever you could activate PRO-WAM while you were using other programs and applications. MISTERED is still available!

DED, FED, and MED all use a similar display screen and strikingly similar commands to enable you to edit any byte of a disk sector, file record, or memory page. This lets you get comfortable with one program yet know how to use all three. The MED display looks like that shown below.

Aside from 4-direction cursor movement, you can find hex or ASCII strings, make changes in hex or ASCII, advance/decrement sector/record/page, insert a NULL



```
 0123456789ABCDEF          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
═══════════════════════════════════════════════════════════════════════════
| b.............. | <00> | 62 00 1F 1C 1F 1E 1F 1E 1F 1F 1E 1F 1E 1F BA F1 |
| ........Level-AR | <10> | 02 00 00 00 00 00 00 00 4C 65 76 65 6C 2D 41 52 |
| dir /app:0.OS 1. | <20> | 64 69 72 20 2F 61 70 70 3A 30 0D 4F 53 20 31 2E |
| 3 => EnhComp #M- | <30> | 33 20 3D 3E 7E 45 6E 68 43 6F 6D 70 20 23 4D 2D |
| 20-072....AS ass | <40> | 32 30 2D 30 37 32 0D 0A 0A 0A 41 53 20 61 73 73 |
| emblers...       | <50> | 65 6D 62 6C 65 72 73 2E 0D 0A 20 20 20 20 20 7F |
| Ordering Inform. | <60> | 4F 72 64 65 72 69 6E 67 20 49 6E 66 6F 72 6D 03 |
| ......../L....1. | <70> | C3 F4 0F 0C B0 05 98 1F 2F 4C C9 F4 0F 0C 31 05 |
| ../L....2.../L.. | <80> | 98 1F 2F 4C C3 F4 0F 0C 32 05 98 1F 2F 4C C9 F4 |
| ..3.../L....4... | <90> | 0F 0C 33 05 98 1F 2F 4C C9 F4 0F 0C 34 05 98 1F |
| /L....5.../L.=.D | <A0> | 2F 4C C9 F4 0F 0C 35 05 98 1F 2F 4C C3 3D 0E 44 |
| A.'.E..=.DB.'.E. | <B0> | 41 14 27 11 45 14 C3 3D 0E 44 42 14 27 11 45 14 |
| .......SunMonTue | <C0> | FF 00 00 00 00 00 00 53 75 6E 4D 6F 6E 54 75 65 |
| WedThuFriSatJanF | <D0> | 57 65 64 54 68 75 46 72 69 53 61 74 4A 61 6E 46 |
| ebMarAprMayJunJu | <E0> | 65 62 4D 61 72 41 70 72 4D 61 79 4A 75 6E 4A 75 |
| lAugSepOctNovDec | <F0> | 6C 41 75 67 53 65 70 4F 63 74 4E 6F 76 44 65 63 |
═══════════════════════════════════════════════════════════════════════════
Banks:  31  Current Bank:   0 Page: X'04' Byte: X'74' => X'B0' =   176

Memory EDitor 1.1 - Copyright 1986 MISOSYS, Inc.
Command [; - @ A B C D F G H I N P Q S X Z]:
```

and push bytes down, quash a byte and pull bytes up, and zap a value to the end of the displayed data. Changes do NOT take effect until you SAVE!

VED lets you edit the current video screen with CARD-type editing. You get block mark and move so you can relocate text on the screen. It combines the facilities of DOSAVE and DOLOAD to save/restore a screen image. With these facilities and the windowing function of EXPORT, you can easily use VED as the "clipboard" facility found on more expensive systems. You can even use it to screen-edit a BASIC program!

TED is a full-screen ASCII text editor that sports a 30K text buffer - while you are using other programs! You get full cursor control, search and replace, insert and overstrike, delete mode, scroll up, scroll down, as well as page up and page down. TED has block mark, block copy, block move, and block delete. Mister ED even comes with an OOPS facility in case you exit TED without saving your text - you can get it back! TED is friendly, fast, and great for those quick editing jobs you need to do while on-line. TED's great for banging out notes when you are right in the middle of a program which you can't interrupt. The LS-DOS 6.3 TED/CMD editor was derived from TED/APP, which came first.

WAMMIES For All

Recollect that PRO-WAM, as released, was originally designed allowing programmers to easily write their own pop-up applications. And they soon started appearing. A good handful of applications were placed on our Compuserve forum as either shareware or public domain programs. In addition, many were also contributed for publication in *The MISOSYS Quarterly*. Here's a summary of the applications which have been made available over the years. Incidentally, with the change of name from PRO-NTO to PRO-WAM, the applications have been affectionately referred to as *wammies*.

CTL255: A filter module by Lynn Sherman to emulate the keyboard matrix accessed by SuperScripsit (which does not use the DOS keyboard driver) so as to enable full use of PRO-WAM with SuperScripsit - including export and import.

DM: A PRO-WAM application by Dick Newman which shows where programs reside on your disk. Works on 40 track single sided double density disks only.

ELEMENTS: A PRO-WAM application by Danny Mullen which pops-up a periodic chart of the elements.

FELSWOOP: A special-purpose PRO-WAM utility by Jeff Joseph used to convert text data for export to either Multiplan or Visicalc.

HELP: A PRO-WAM application by Matthew Reed which accesses a DOS-type HELP file to pop-up help screens similar to HELP/CMD.

LIST: A file lister module by Ken Kroninger for PRO-WAM.

MAPMEM: A PRO-WAM application by Don Brandt used to display the contents of selected low/high memory modules and system data. It also calculates the machine size (RAM banks).

MODEM: A PRO-WAM application by Don Brandt that supports XMODEM file transfers. It requires that *CL be set to COM/DVR.

MXDUMP: A PRO-WAM application by Gregg L. Rupple to screen dump the Model 4 screen with lo-res graphic characters.

NXWAM: A printer setup PRO-WAM application by Danny Mullen for the Star NX-10 printer.

OKIDAT: This is an application program by Mark Bowman designed to run with PRO-WAM to send initialization strings to your Okidata printer.

PAW: A utility by Roy Soltoff to enable a single keystroke execution of any PRO-WAM application directly from a PRO-WAM library file.

PG2: A small PRO-WAM application by Jacques Verhelst which is used to page through either of two data files: DATA1/TAB and DATA2/TAB.

PRCTRL: A PRO-WAM application by Paul Bradshaw which can be customized to send control strings to any printer. It features user-definable keywords, user-definable code sequences, substitutions, repetitions, and more.

SVC: A PRO-WAM application by Doug Tittle used primarily by programmers to pop-up a database of DOS service calls. By popular demand, it includes a Search by SVC Name and Number ability.

WAMDUMP: A utility by Claude Hunter to convert a PRO-WAM core-image application file into a standard load-module image which can then be disassembled by PRO-DUCE.

WINCALC: A programmer's calculator by Bryan Headley which provides all the mathematical and logical operations a Z-80 programmer may require during the development and maintenance of assembly language programs.

It took a few years for folks to migrate to the new name, PRO-WAM. But it eventually took effect. One good way of assisting in the changing of a product's name is to release a new version. Based on the initial feedback from PRO-WAM Version 1 users, MISOSYS continued to add features enhancing the value of the product. In the Spring of 1987, *The MISOSYS Quarterly*

announced the release of version 2. This announcement conveyed a good deal of information concerning the utility of PRO-WAM; thus, it makes sense to provide that information again in this article. Many current TRS-80 users have not had access to TMQ issue I.iv.

Enter PRO-WAM Release 2 [Spring 1987]

Since the release of LS-DOS 6.3, we have had many requests from PRO-WAM (nee PRO-NTO) users asking for revisions to the BRINGUP application to extend its date support beyond 1987. Well here is an announcement that should please every PRO-WAM user; and make many other folks run out to get PRO-WAM Release 2.

In the *80 Micro* review of PRO-NTO Release 1 back in November 1985, Hardin Brothers said, "PRO-NTO is one of the most useful products for the Model 4 I've seen." Well we have supercharged this new release to include just about every feature requested by our PRO-WAM users and added other significant enhancements that should make PRO-WAM even more useful to just about every owner of a 128K Model 4.

To those Model 4 owners uneducated about PRO-WAM, here's a little PRO-WAM overview. The "WAM" stands for "window application manager". PRO-WAM is a manager of applications which pop up in windows on your video screen. You can pop up a PRO-WAM application whenever you are running any other "DOS-behaved" program. The PRO-WAM application can capture anything on the video screen by means of a function called "import". Similarly, a function called "export" can be used to pass back to the interrupted program, anything which is in the PRO-WAM application's window.

There are essentially two parts to PRO-WAM: (1) a resident module which supervises and controls the operation of the video windowing and application execu-

continued on page 38

PRO-NTO: TRSDOS 6.X's Sidekick

by John B. Harrell III

*****	
PRO-NTO runs on the Model 4/4P (128K) and requires two disk drives. MISOSYS, Inc., P.O. Box 239, Sterling, VA 20167, 703-450-4181	
Easy to use:	*****
Good docs:	*****
Bug free:	***
Does the job:	*****

Finally, TRSDOS 6.X users have a program comparable to Borland's MS-DOS Sidekick and Montezuma Micro's CP/M Monte's Window. Like these packages, PRO-NTO gives you a memory-resident calendar, calculator, index-card filer, and phone dialer at the touch of a pair of keys, even from within another application. In addition, you can link your own applications to PRO-NTO's windows. All in all, PRO-NTO is one of the most useful products for the Model 4 I've seen.

How it Works

PRO-NTO resides in high memory and reserves a 32K RAM bank for its programs and various buffers. When you call up PRO-NTO, it saves your current screen and the DOS overlay library in one of these buffers.

You can call up to four windows simultaneously. though you can work only in the most recently called window. When you finish using PRO-NTO, it closes any open window, restores the previous screen's contents and the DOS overlay, and returns control to your interrupted program.

One of PRO-NTO's notable features is its system of data transfer between windows and applications, referred to as data import and export. PRO-NTO lets you import data into a window from a previous screen or export data from the current screen to another application. For example, I used PRO-NTO to generate a small job control language (JCL) file with the note card application. When I finished writing it, I used the export capability to feed the text to the DOS Build command. This very capably overcomes TRSDOS's lack of a text editor.

The Applications

When you type In PRONTO at TRSDOS Ready. the program loads itself, links into your system, and installs a keyboard filter. PRO-NTO then loads its four default applications (the calendar, calculator, index-card filer, and phone dialer) into the alternate RAM bank, along with a DOS library and a user-defined universal application, and returns control to DOS. You can change the default applications to any of PRO-NTO's seven other applications with a supplied utility.

Pressing control-P activates PRONTO - it instantly lists the installed applications. I'll describe all 11 of them below.

Address maintains an index-card file of addresses you use like a Rolodex file or as a mailing list (the data structure matches that of Powermail Plus).

Address accommodates all standard address-file entries, like names and addresses, and adds a few new ones. In addition to two-line street addresses, you can include a company name and nine-digit zip codes.

Address provides two other fields for key words (useful for sorts) and telephone numbers. It also offers up to 24 flags you can set or reset for use as search criteria. For example. you could set flag 6 for customers who get a 10 percent discount on merchandise.

BRINGUP, a reminder file and appointment book, lets you schedule up to 12 entries for each day in priority/time order. The only limit to the number of entries you can store is the amount of disk space available. BRINGUP represents the only PRO-NTO application that doesn't use MISOSYS's standard data base format.

BRINGUP defaults to the system date on initial display. If you invoke it from the calendar, it defaults to the date currently selected. Unfortunately, PRONTO doesn't highlight dates on which you've scheduled activities.

PRO-NTO's perpetual calendar displays the month for any year from 1582 to 4902. PRO-NTO highlights the current date with a pair of flashing angle brackets. You can pass this date to the BRINGUP mode if you invoke it from the calendar.

PRO-NTO's floating-point calculator provides a rudimentary four-function capability with a one-number memory. You make entries to the calculator using algebraic notation. PRO-NTO considers all operators of equal precedence and calculates the formula from left to right in the sequence entered.

PRO-NTO displays results in decimal or scientific notation. If a number exceeds the decimal limits, PRO-NTO displays it in scientific notation (for example, it displays 0.00000002567 as 2.567000E-8). The calculator works with single-precision floating-point numbers and maintains six or seven decimal digits of precision.

The reverse Polish notation (RPN,) four-function calculator is really a programmer's tool and provides for data entry in binary, octal, decimal, and/or hexadecimal format. You can use three additional operations: logical And and Or, and Exclusive Or. While the calculator doesn't support negation and complements, you can quickly calculate these using simple arithmetic operations.

Unlike the floating-point calculator. the RPN calculator lets you enter an operation sequence as a string and you can edit this prior to pressing the enter key. You have to separate operators and operands by a space. The default result base is decimal and you can change this to any other base by appending the appropriate designation to the entire expression.

The card filer and notepad [Card) provides a convenient way to store individual notes of 12 lines by 40 characters. PRO-NTO lets you identify a Card file with an eight-character key: it automatically appends the date and time to the record.

Card supports commands for adding, deleting, and finding records in the notepad. You can move forward and backward through the data base with single-key commands. You can select and print a single card or all the cards in the data base. Also, a simple editor lets you make changes to the note.

CHARSET displays all 256 ASCII characters in a single window. When you position the cursor on a character, CHARSET displays the character's hexadecimal and decimal numbers. You can toggle to the alternate character set (characters in the hexadecimal range COH-FFH) with a simple keystroke.

The best feature of this application is the ability to build a string and then export it back to a waiting application. For example. you might build a graphics string by locating the appropriate character(s) in CHARSET with the arrow keys and pressing the enter key for each character. When you finish, you can transfer the generated string to the waiting program.

The telephone list and dialer application (Dialer) is a small data base of phone numbers and descriptions. Dialer fully supports the Hayes SmartModem and compatibles. You can dial numbers directly from the data base or manually enter a number for Dialer to call. You ran also export a number from the address (or any other) application.

The storage field for the actual telephone number is only 16 characters long, but this isn't really a limitation for those services that require more numbers; Dialer supports 15 user-defined macro keys you can use to store up to 16 characters in the original phone number.

The DOSAVE application saves the contents of the previous screen to a disk file as 24 strings of 80 characters each. This application has limited use and is provided in source form as an example of how to write an application for PRO-NTO.

The terminal communications program (Term) is another example of a simple application written for use with PRO-NTO. The only filtering performed converts a form feed signal to the traditional .Screen Clear command. Term won't run unless you install the *CL device with COM/DVR.

Typer jerry-rigs your keyboard and printer so that the printer types out the character you press on the keyboard. It uses a two-line window so you can type a line at a time directly to the printer. Here again, this program seems more a demonstration of PRO-NTO than a practical application.

In addition to the four default applications PRO-NTO loads on boot- up, it loads two other programs, the Library Executive (LIBEXEC) and the Universal Application Invoker. LIBEXEC lets you access all TRSDOS [library] commands. You can also run other programs with the Run command if you know that it will run in the DOS overlay area.

The Universal Application Invoker runs any PRO-NTO application from disk, since you can configure PRO-NTO for only four of the supplied applications. This gives you access to PRO-NTO's other programs.

The Utilities

PRO-NTO Includes three utilities. PRUN lets you bypass PRO-NTO's program menu so you can call an application much as you would from DOS. Instead of answering prompts to invoke an application, you call a program by typing in PRUN FILE NAME from DOS. You can also use PRUN in an autoexecute file to automatically load one of PRO-NTO's programs on boot-up.

PSort, a sort function written specifically for PRO-NTO data base files. can be easily used for other data structures. Misosys has provided complete documentation on the data structure required for this utility.

MISOSYS also includes a full help utility. It provides documentation for each of PRO-NTO's features and is similar to the TRSDOS 6.2 Help command.

Blemishes

PRO-NTO's major strength is also its weakness: The generic nature of the program means that it won't work with several software packages, such as SuperScripsit and LeScript. Both of these programs ignore PRO-NTO because they don't read characters from the standard TRSDOS keyboard driver (*KI).

Also, I found that having only four PRO-NTO applications in the RAM bank just wasn't enough - I continually referenced the program disk for other applications. Even after I customized the default set-up (see fig. 2a), I needed the PRO-NTO disk handy when I wanted an uninstalled application. Since. I was using only one of my alternate 32K RAM banks, I thought I could put the other to use. Figure 2b shows a JCL procedure that loads PRO-NTO and copies specified applications and data bases to Memdisk. You must format Memdisk for single-density operation to keep the granule size as small as possible and to minimize wasted space.

While DOSAVE preserves the contents of the video screen, you can't simply restore the screen by listing the contents. Each 80-character line ends with a carriage return and this makes the display double-space. Also. writing to the last location on the screen (lower right corner) scrolls the entire screen.

PRO-NTO is an excellent tool. Not only does it have applications immediately available for use, but the large amount of technical information provided makes it easy for programmers to use it for other applications. And the documentation (90 pages) is clear and succinct. Disregarding the few problems mentioned. this package is truly outstanding.

Figures 2 (a) and (b)

```
(a)     1. Calculator
        2. Calendar
        3. Card
        4. Address

(b)     PRO-NTO
        SYSTEM (DRIVE=2,
DRIVER="MEMDISK")
        C
        S
        Y
        BACKUP/DAT:0 :2
        COPY BRINGUP/APP:0 :2
        COPY CHARSET/APP:0 :2
        COPY DIAL/APP:0 :2
        COPY RPNCALC/APP:0 :2
        COPY DOSAVE/APP:0 :2
        COPY TERM/APP:0 :2
```

Figure 2a Custom PRO-NTO default setup

Figure 2b. TRSDOS 6.2 JCL file that installs a modified PRO-NTO and loads the other 32K RAM bank with additional applications and data bases. The responses to Memdisk concern the second RAM bank and single-density operation.

PRO-WAM: continued from page 36

tion, and (2) a set of application programs some of which may be made to reside in memory. PRO-WAM uses about 2500 bytes of your machine's high memory and one memory bank. That's why it requires a 128K machine.

Application programs provided with PRO-WAM include a calendar, a bringup and tickler file, an address (Rolodex™-like) file, a 3x5 card file, a modem autodialer, a 4-function calculator, a disk-based keystroke-multiplier, a todo list manager, a terminal program, and more.

Let's take a look at the features which have been added to the window manager.

PRO-WAM now supports Model 4 reverse video detection and restoral at each level of application execution (there are four levels). For those of you using Visicalc or Multiplan, you can now get into PRO-WAM applications and return to your spreadsheet screen with reverse video intact. Besides that, PRO-WAM applications can make use of reverse video in their screens; we use it in CARDX.

More Model 4 folks are taking advantage of extended memory add-in boards to gain additional memory banks. So we added a BANK parameter for PRO-WAM installation to force PRO-WAM to use your designated bank number rather than automatically search for a spare. Even if you don't want to make use of that, PRO-WAM's automatic search will now look up to bank 30.

PRO-WAM is installed as a keyboard device filter. Under release 2, "PRO-WAM (OFF)" now unfilters the *WM device rather than RESET *KI. This leaves your *KI device chain intact (except for the PRO-WAM removal).

PRO-WAM now accesses application modules from a library - not from standalone modules. This keeps the apps from cluttering up your directory. PRO-WAM initially loads from the WAM0/

APL library. Thereafter, the UNIVERSAL function request for "Application?" is now satisfied from that library on the drive where the library was found unless overridden by the application module specification (mspec$) entry. Even though a library can hold 32 applications, more than most folks will have use for, UNIVERSAL can still specify the library to access or default to WAM0/APL. Specify "modname" to access WAM0/APL for "modname". Specify "#modname" to access WAM#/APL for "modname". The new drive will be again used in subsequent UNIVERSAL requests until changed by explicit entries in a mspec$ (either UNIVERSAL or programmed execution). You can have up to ten application libraries (designated WAM0/APL through WAM9/APL).

We've expanded the capabilities of the window supervisor call with two significant enhancements: programmed control of export and programmed invocation of an application module. We use the former to automate the PHRASE application and the latter to enable the "bringup hot key" in CAL. Of course both facilities are available to every programmer, as well.

To expand on the concept of "protected fields", we've added the concept of "protected characters" to the behavior of export/import. Any character within the defined rectangular export/import area which has a value greater than 127D (i.e. bit-7 set) will not be output/input; i.e. it will be considered a protected character. Most typically this would be a reverse video character; however if reverse video is not in effect, any graphic or special character greater than 127 will be considered protected and not output/input. This will be extremely useful with our revised CARDX card file application.

Export/import now supports a logical ENTER character which will be translated to a hard carriage return regardless of the import/export state (CR or NOCR). Thus, a text phrase can be export as "multiline" even though it is a continuous stream of characters without a carriage return. The logical ENTER character is tested before the protected character test; thus, values greater than 127 are acceptable for use as a logical ENTER. The default logical ENTER character is 127d (7FH) which can be entered by <CLEAR><SHIFT><ENTER>. We've added an "ENTER" parameter to the PROWAM installation so you can change it to one of your choice. It's even configurable with DEFAULTS.

We've used the expanded features in the application manager to make considerable improvement in the existing applications and added some powerful new ones. Here's what's been done in the application arena.

Many users requested that the CALendar application flag days which have a BRINGUP activity entered. Well we added a facility to display an asterisk in the leftmost position of a day cell if there is an active BRINGUP record for that day. CAL scans the entire BRINGUP/DAT file and maintains a table of flags indicative of activity per day for an entire year. The file is only re-scanned when you change years (if it found a BRINGUP file initially). We also added a "bringup hot key" in the CAL command list so that you can easily position to the day showing an asterisk and hit one key to bring up the list of activities you have scheduled for that day. That facility has become indispensable here at MISOSYS.

Speaking of BRINGUP, we altered the structure of the data file so that the year spans 16 years: from 1984 to 1999. The date field storage was also reorganized so that is can be sorted in ascending order. Since sorting by PSORT requires a data file header record, that was added too; BRINGUP data files can now be sorted by date and time using PSORT. This requires a convert utility for your existing files which is included with the release.

Now sorting the bringup data file only makes sense when you have a facility for printing activities. Enter the new BUP/APP application. This wammie, which can be invoked directly from BRINGUP, supports printing of bringup activities by date range using a syntax exactly like that in your DOS for BACKUP and DIR date parameters. We also added '.' as a shorthand for "today". Standard user-input print formatting is supported similar to that facility in ADDRESS and CARD. BUP also incorporates the REMOVE command which used to be in BRINGUP.

I keep an ADDRESS file in the computer on my desk. Since I like to use it to capture address headings when writing letters, I got tired of exporting the address record as it appeared in the window which required editing in my text editor. That's why I designed HEAD. This new application accesses the ADDRESS/DAT file and presents an address record in the format:

Company
First Last
address1
[address2]
City, ST ZIP

If the Company field is blank, then it won't appear in the window. Likewise, if Address2 is blank, it won't take up a line. I even coded HEAD to drop trailing spaces from the fields so, for instance, there's no big gap between First and Last. HEAD provides commands just like ADDRESS: search, next, prev, first, last. An <ENTER> will automatically export the record. The application is useful for extracting address information for use as a heading in a letter. It sure saves me some time and eliminates needless editing.

We've modified ADDRESS/APP to pick up the sort key and key length so that the search command now uses the external key information specified in the data file header record rather than hard coding the search to the lastlfirst fields. This means you can alter the key location and length fields in the ADDRESS/DAT file to order/search by ZIP, Company, or whatever. This was to satisfy a few requests from folks wanting ADDRESS to sort and search on the Company name field.

I fail to understand why the MS-DOS world clamors for key-stroke-multiplica-

tion facilities such as SUPERKEY, POWERKEY, and whatnot, yet the TRS-80 users rarely explore the capabilities of their own KSM facility which has been available in LDOS and DOS 6 for years. Nevertheless, we have added a new application called PHRASE. This wammie accesses a PHRASE/TXT file consisting of any number of phrases. You can scroll through the phrases or enter a phrase mnemonic. An <ENTER> will export the phrase selected. Your phrases can make use of the logical ENTER character of export. This is akin to a more or less infinite KSM. The syntax of a phrase is a 2-character mnemonic followed by 1-78 text characters and terminated by an <ENTER>. The PHRASE/TXT file is composed of any number of phrases. It is terminated by a period, '.', following the last <ENTER>. Use any ASCII text editor to input and edit the PHRASE/TXT file (use TED, for instance).

The phrase facility can work wonders when you want to prepare letters, drafts, etc., using "canned" phrases. You can also use PHRASE to store strings for application print formats. How about DOS command strings? It boggles the mind!

CARDX/APP is a new application which is an enhanced CARD file. It supports up to ten CARDx/DAT files [0-9] rather than the single one under Release 1. It makes use of reverse video to designate protected screen positions which cannot be edited (nor exported). CARDX will actually skip over any screen position which is in reverse video.

CARDXF/APP is used to construct and edit a CARDFORM record for use with CARDX. This wammie allows you to create new data files and add/edit a cardx record identified as "CARDFORM". Reverse video, which is used for protected fields in CARDX/APP, is toggled via <CTRL-R>. The screen indicates the current state of reverse video. CARDXF also allows you to "populate" the CARDX file with any designated quantity of records initialized with the "CARDFORM" form. It's easy to use the cardxf editor to "paint" your form onto the text area of the window. By using reverse video for every position which you want "protected" from entry while using CARDX, you can create a form to be filled in later. When using CARDX, the cursor will be automatically moved to the unprotected positions. This combination of CARDX/CARDXF can really provide you data base functions.

Around here we always seem to forget to do this or forget to do that. Sometimes BRINGUP doesn't work when you just want to keep a list of items to be done. Remember that todo pad? That's why I designed TODO/APP, a new application which is used to establish and maintain a simple list of items "to do". Now I wanted to keep Brenda's todo list together with mine so I designed it with a field to indicate "who does it". This wammie generates a TODO/DAT file which is similar to a BRINGUP file. Allows priority of 1-8, a "who does it" value of 0-15 where a value of 0 is considered global (more on that later), and a text field of 29 characters. The "add" date is added to the record. Commands allow you to "add" an item, mark an item as "done" which deletes it, "file" any changes, move to "next" or "previous" display page, and "list" items. A page displays the 8 items in a disk record; the display works like DIALER). The list command allows you to print all records or the current record additionally specifying either all "who does it" or a particular "who does it". Any item tagged as "who=0" will always appear on a listing. Forget Hi's "job jar"; TODO does it!

PRO-WAM comes with an on-line help facility which has been updated to include all of the features added in release 2. It can keep you from having to resort to the user manual when you forget a particular operation of PRO-WAM. Of course, since every application has a command menu, you will rarely have to resort to the manual once you have familiarized yourself with the operation of the applications you will use.

PRO-WAM also comes with some improved utilities; and some new ones to boot. With all the talk about sorting BRINGUP files by date and time, you may have realized that PSORT needed some work. Well PSORT has been revised to accept an expanded data definition of up to two key sort fields. Not only that, but the fields can be either character or integer.

Because of the switch to application libraries, we've included WAMLIB, a utility to build and maintain application libraries. WAMLIB has functions to create a library as well as the functions:

- add a module
- delete a module
- replace a module
- extract a module
- list module directory

WAMLIB is menu driven and easy to use.

Since applications are now invoked from libraries, PRUN has been enhanced to load applications from libraries, too. However, since some folks may have a need to invoke an individual application file, PRUN has been enhanced with a parameter APP (abbreviated A). PRUN will normally load the application from the WAM0/APL library (or other library when the module name is prefixed with the library number). If you want to force PRUN to load a standalone application (i.e. for testing before adding it to a library), specify the APP parameter as in: PRUN application (APP).

Well that's it. PRO-WAM release 2 includes the window application manager, 18 different applications in a library, a library manager, the PSORT utility, the PRUN facility, a help facility, and a DEFAULTS utility to customize your PRO-WAM configuration; all on disk.

TESTIMONIALS

Over the years, we have received various questions and responses referencing PRO-WAM. Here's some excerpts of past issues of *The MISOSYS Quarterly* which may shed further light on how PRO-WAM

can benefit you.

Harry Hopkins wrote, "I think I know what PRO-WAM is and can do, but can someone make me a clear, 1000 words or less explanation?"

Tom Gallaudet chimed in with an unsolicited testimony as to how he put PRO-WAM to use. "PRO-WAM is one of those programs that sounds neat in the magazine ads. Well, I received it, installed it and played a little and then decided that it was a really neat program but I wasn't sure what I wanted to do with it so it sat for a while. After a couple of months and several ad 'rereads', I started to play with it again and now I could not live without it! After pressing two 'key combinations' whatever program (if any) you were running, stops and a little window appears in (or close to) the middle of your screen. If you hit the break key, the window goes away and your program continues where you left off never missing a beat.

I have an auto dial modem and use the DIALER all the time, I use the ADDRESS for anyone that I send any info to and code each entry for the type of info that I send so I can quickly print a list of who got what. I have the BRINGUP appointment tickler automatically run when I boot the system so that I'm forced to see what I am supposed to do each day.

It also has a CARD program which is a little text file which I use for storing notes on people like topics I want to discuss with my partner when we call. No matter what program I'm running, I can immediately call up his 'CARD' and have the list. Besides what comes with the main program, there are several additional public domain programs that work with it.

PRO-WAM uses some high memory and all of the first 32k partition leaving the second 32K bank for a memdisk to store the most used PRO-WAM data files or as the memory for the TEXT Editor. It is a very well written program and it will interface with every program I have tried with only one problem that I could complain about and I haven't complained because it's not that big a deal. The problem is: If you are using RS's Videotex Plus and you want to use the TEXT Editor, it's reset button time. Videotex just freezes solid. But you can use all the other programs with no ill effects. The bottom line is: it takes a while to figure out how to use it but once you get started the possibilities seem endless. It's very well written and well worth the price. (And it's worth more then 1000 words!) And I also got the MR. ED add-on package which has TEXT, FILE, DISK, MEMORY, and VIDEO EDITORS."

Ken Kane wrote, "My favorite new application for PHRASE/APP in PRO-WAM is for browsing the DL's. I set up a KSM key to invoke PHRASE as per Roy's instructions on pg. 112. Last night I did a search for ink jet printer reviews on several SIGs. My PHRASE/TXT file included:

```
S1BRO/KEY:DICONIX<±>
S2BRO/KEY:HEWLETT<±>
S3BRO/KEY:INKJET<±>
S4BRO/KEY:JET<±>
```

I didn't find much, but was my searching fast! I searched about a dozen DL's on several SIGs in about 2-3 minutes. True, I could have used separate KSM keys for each BRO/KEY: but this is less disruptive. I used to keep two keys tied up under KSM to log onto CIS at 1200 to download, and the other at 300 baud et alia for XMODEM. I now have other ways to use my DTERM communications program including experiments logging on at 2400 via TELENET, and at 2400 to a local BBS. Well, all these different baud rates and script files are stored in PHRASE/TXT, very accessible yet very easily changed. PHRASE/APP has shortened the export function by several keystrokes.

Since the first PRO-WAM release I have used CARD/APP to keep track of message numbers to reply to, (composing replies off-line, etc). I have always loved KSM functions, being a better button-pusher than typist. PHRASE/APP greatly expands the KSM function only a little less accessible. But very easy to change and to expand. One more reason to hang in there with the ol' Model 4."

Michael Rogers was so thrilled with PRO-WAM, he wrote, "Any Model 4 user reading this who does not own PRO-WAM is missing out on an invaluable utility/application which is both a great time saver and a program which could be used to revitalize interest in your Model 4, should it be flagging. All users should appreciate its facilities and the imaginative will find a myriad of applications. I have recently used its brilliant export-import facility to save retyping a database I wished to transport from one program to another. Besides reducing typing, it was fun - giving that satisfying, 'my computer is saving me time' feeling."

Here's some examples for using the EXPORT capability of PRO-WAM which I used to clarify the situation with a user. Note what this power can do for you!

Here is an example for ALLWRITE. You are in ALLWRITE preparing to type a letter and positioned at where the heading is to go. You want to send the letter to someone whose address is in your ADDRESS file. So you pop up PROWAM via <CTRL-P> then invoke UNIVERSAL via <F3> and type "head" in response to the "Application ?" prompt. You then issue a SEARCH for "SMITH", find the right one, then press <ENTER>. the export is done automatically and you are back in ALLWRITE seeing the heading being brought into your text as if you typed it yourself. That's quite easy, but since HEAD controlled most of the export, you didn't get the flavor of it as you should.

Here's another. You previously jotted down a note to Smith in a CARD file record the other day while you were running some other program. Now you want to bring that note into the letter to Smith without re-typing the note. So you have the ALLWRITE cursor at the place for text insertion and you are in insert mode.

Press <CTRL-P>; select the CARD file, search for the note, then press <CLEAR-RIGHT-ARROW>. You will see a cursor at the upper left corner of the window. You move this around via the ARROW keys so that it is positioned at the upper left corner of the text (text must be envisioned as a rectangular region). When you have the cursor in the right spot, press <CTRL-B> to mark the beginning of the block. Then move the cursor via the ARROW keys to the lower right corner of the text block that you wish to export. You can choose to depress <ENTER> at this point or <SHIFT ENTER>. The difference is that <ENTER> will export line by line but an <ENTER> is added after the last character of each line. If you use <SHIFT ENTER> to close the text block, the marked lines will be export as if they were one continuous stream of characters. I suspect that this would be better for ALLWRITE - unless the text were a table. In any event, when you close the block, the CARD window will go away and you will be back in ALLWRITE with the marked CARD text magically piped into your letter. Can these examples illustrate a time and finger saving operation for you?

Lastly, Michael Rogers reports, "I have used CARDX to create pop up help screens for BASIC. Besides having a help window which in no way interferes with the BASIC program you are writing, PRO-WAM's import/export feature allows examples from the help windows to be transferred directly into the program you are writing. I have found this particularly useful when using seldom used statements and functions. Instead of digging through the manual, CARDX's search function allows quick access to the information I need and often the example field on the card provides me instantly with program material. Placed on a ramdisk, a search for an item on the last card in a file of over 140 cards (extended memory beyond 128K recommended) takes only about 5 seconds. Also all references to a particular statement, keyword, or function are easily found, often more efficiently than by using the existing index in the manual. In the past, I rarely used help files but the interactive nature of PRO-WAM certainly made the effort of creating my BASIC help cards very worthwhile.

I have also used PRO-WAM to create pop up help files for 'LeScript'. Those familiar with this word processing program know that it has an alternate screen for help files. However, if you want to use the alternate screen to edit two text files at once (two copies of the same file in memory at once can be handy), you lose access to the help files. PRO-WAM allows you to have pop up help files and have two text files on alternate screens. Making these help windows was fairly easy, using import to transfer the existing 'LeScript' help file data onto PRO-WAM cards."

THE WRAPUP

If you have kept your eye peeled to the MS-DOS market, you have noticed that a relatively new popular class of program applications is the Personal Information Manager, or *PIM* for short. These tend to become computerized dossiers which tend to assist a person - business or otherwise - in carrying out their day-to-day operation. For instance, efficiency managers will always recommend a person to create a list of what needs to get done. Things won't always occur unless they're on a list. Classically, such a list was called a *todo* list, things that you needed to do. When you deal frequently with a set of other individuals or business contacts, it helps to organize them into a telephone or address list (for mailing purposes). You also take notes from time to time which may very well relate to the contacts in your address or telephone list. Or perhaps your notes tend to become more organized into a small base of data. Lastly, if you don't keep track of your appointments, you can easily forget one or two. Appointment schedulers, i.e. calendar programs are the salvation of forgetfulness - all you need to remember is to use them.

Now all of the kinds of personal information management previously noted are handled by PIMs. From the description and utility of PRO-WAM contained in this article, it is not difficult to see the resemblance PRO-WAM has to a Personal Information Manager. Such is the case, for PIMs evolved from the Sidekick-type desktop managers. What stands the PIM apart from the Desktop Manager is an integrated facility to create a printed page containing everything a person needs to deal with on a particular day - or a week. Such a printout is typically used by a salesperson travelling without a usable laptop computer. The integrated facility scans all of the associated files and bring together on one printout, the appointments, telephone records of those contacts on the appoint list, pertinent notes which have been made, etc. This is not difficult to accomplish. The CALA4/BAS program noted above ties together the PRO-WAM appointment application (BRINGUP) with a monthly calendar. If one were consistent with a key field to connect associated records, it would not take a great deal of programming to develop the integrated facility needed to produce the single printed page. Is that a challenge left to our readers?

Anyone still using a Model 4 which has 128K of RAM that doesn't have PRO-WAM in their library of programs is missing out on one of the best programs ever developed for the Model 4. There's still time to grab this one and put it to use. See our special announcements in this issue of *The MISOSYS Quarterly*.

References:

The MS-DOS Encyclopedia, by Microsoft Press.
Computer Systems Architecture, by Jean-Loup Baer, Computer Science Press.
Grolier Encyclopedia of Knowledge, by Grolier Inc.
Microcomputer Architecture and Programming, by John F. Wakerly, John Wiley & Sons.
The MISOSYS Quarterly, various issues.

1528 Wiggins Ave.
Saskatoon, SK S7H 2J8
CANADA

HELP to TEXT & Back

By Brian Davis

I wish to thank Roy Soltoff and MISOSYS for their continued support of the Model 4 and those who still use it.

INTRODUCTION:

The following programs were written to convert the LSDOS 6.3 (and TRSDOS 6.2) DOS HELP file to text and then back into "Help" format in order to correct errors in the file and enhance some of the command explanations. The programs can also be used to generate other help files for any other application.

I find that the advantage of the DOS help files is that each contains a directory of its contents which allows quick access to the desired information.

I could not compile these programs using Tandy's BASCOM compiler since it will not handle random access files with record numbers greater than 32767. Therefore be prepared for slow conversion (especially by TXT2HLP/BAS).

Hardware Requirements:

Model 4/4D,64K,1 drive

DOS "HELP/CMD" Program Requirements:

To understand the programs, it is first necessary to understand the requirements of the DOS HELP/CMD program. The text which follows explains some of the requirements. The text is included on DISK NOTES in the proper format to be converted into "Help" format.

The help file source format is as follows:

(1) The first line must be "<D>" to identify the first directory entry.

```
10 REM ——— HLP2TXT/BAS
20 CLS: PRINT CHR$(16);SPACE$(35);"HLP2TXT/
   BAS";SPACE$(34);CHR$(17);
30 PRINT SPACE$(15);"Convert a file from DOS HELP format to
   TEXT format";SPACE$(15);: PRINT
40 PRINT SPACE$(14);
   "————————————————————————————"
50 PRINT SPACE$(14);"|     Authored By Brian Davis, July
   19, 1988       |"
60 PRINT SPACE$(14);"|                    1528 Wiggins Ave.|"
70 PRINT SPACE$(14);"|                    Saskatoon, Sask. |"
80 PRINT SPACE$(14);"|                    Canada S7H 2J8   |"
90 PRINT SPACE$(14);"|                                     |"
100 PRINT SPACE$(14);"| This program carries no warranty of
   any kind and |"
110 PRINT SPACE$(14);"| the author assumes no liability for
   direct or    |"
120 PRINT SPACE$(14);"| incidental damages as a result of
   its use. It    |"
130 PRINT SPACE$(14);"| may be copied and distributed
   freely as long as  |"
140 PRINT SPACE$(14);"| the author identification remains
   intact.        |"
150 PRINT SPACE$(14);
   "————————————————————————————"
160 PRINT
170 REM ——— VARIABLE INITIALIZATION
180 HLPFILE$="": REM — name of input help file
190 TXTFILE$="": REM — name of output file in text format
200 RECNO = 0: REM — the record number of the help file
210 REVERSE=0: REM — a flag for reverse video, 0=off, 1=on
220 EXPANDED=0: REM — flag for character expanded with
   space,0=not,1=expanded
230 H$="": REM — one character buffer for help file data
240 I=0: REM — general counter
250 DIRMAX=0: REM — maximum number of directory entries
   found in help file
260 DIM DIRTXT$(260): REM — array of 260 directory text
   entries starting at 0
270 DIM DIRADD(260,2): REM — array of high/low directory
   address bytes
280 HI=0: REM — array location for high order byte (for
   program readability)
290 LOW=1: REM — array location of low order address byte
300 DIRSTART=0: REM — the starting address of the directory
   section.
310 REM — This is the last two bytes in the help file.
320 MAXRECNO=0: REM  Maximum record number in source help file
330 REM ——— GET FILENAMES
340 REM — The input filename is a random access file and
   has a logical
350 REM — record length of 1. The output filename is a
   standard text
360 REM — file and has a logical record length of 256.
```

```
370 ON ERROR GOTO 960
380 INPUT "Enter INPUT HELP FILENAME
   (prgname/HLP.password:d) - ",HLPFILE$
390 OPEN "I",1,HLPFILE$: CLOSE 1: GOTO 420:
   REM - If this fails, the RESUME
400  REM - will print the error
410 PRINT "File not found!": GOTO 380
420 OPEN "R",1,HLPFILE$,1
430 FIELD 1,1 AS H$
440 INPUT "Enter OUTPUT TEXT FILENAME
   (prgname/ext.password:d) - ",TXTFILE$
450 OPEN "I",2,TXTFILE$: CLOSE 2: INPUT
   "File exists!...Overwrite?(Y or N) -
   ",YN$: IF YN$="N" OR YN$="n" THEN PRINT:
   GOTO 440
460 REM - If the file does not exist, the
   RESUME will continue
470 ON ERROR GOTO 0
480 OPEN "O",2,TXTFILE$: REM - Let default
   to LRL of 256, BASIC will not
490  REM - allow 256 to be specified.
500 REM - LOAD DIRECTORY INTO ARRAY
510 CLS
520 PRINT "READING DIRECTORY ....": PRINT
530 MAXRECNO = LOF(1): IF MAXRECNO<0 THEN
   MAXRECNO = 65536!+MAXRECNO
540 REM - LOF goes negative after 32767
550 GET 1,MAXRECNO
560 DIRSTART = ASC(H$)*256
570 GET 1,MAXRECNO-1
580 DIRSTART = DIRSTART + ASC(H$)
590 RECNO = DIRSTART: GOSUB 920: REM - Get
   character from file
600 WHILE RECNO < MAXRECNO-2
610    IF ASC(H$)<127 THEN
   DIRTXT$(DIRMAX+1)= DIRTXT$(DIRMAX+1)+H$:
   GOSUB 920: GOTO 720
620    DIRMAX = DIRMAX + 1
630    REM - DIRMAX is used as a general
   counter until it becomes the maximum
640    REM - directory number
650    DIRTXT$(DIRMAX)=DIRTXT$(DIRMAX)+
   CHR$(ASC(H$)-128): GOSUB 920
660    DIRADD(DIRMAX,LOW)=ASC(H$): GOSUB 920
670    DIRADD(DIRMAX,HI)=ASC(H$): GOSUB 920
680    PRINT "ADDRESS = ";
690    IF LEN(HEX$(DIRADD(DIRMAX,HI)))<2
   THEN PRINT "0";HEX$(DIRADD(DIRMAX,HI));
   ELSE PRINT HEX$(DIRADD(DIRMAX,HI));
700    IF LEN(HEX$(DIRADD(DIRMAX,LOW)))<2
   THEN PRINT "0";HEX$(DIRADD(DIRMAX,LOW));
   ELSE PRINT HEX$(DIRADD(DIRMAX,LOW));
710    PRINT "  ENTRY = ";DIRTXT$(DIRMAX)
720 WEND
730 REM - GET TEXT AND CONVERT
740 CLS
750 PRINT "REFORMATING HELP FILE ENTRIES...
760 FOR I=1 TO DIRMAX
770    RECNO=(DIRADD(I,HI)*256) +
   DIRADD(I,LOW) + 1
780    GET 1,RECNO
790    PRINT #2,"<D>": PRINT "<D>"
800    PRINT #2, DIRTXT$(I):PRINT DIRTXT$(I)
810    WHILE H$<>CHR$(12)
820       IF ASC(H$)<127 THEN PRINT #2,H$;:
   PRINT H$;: EXPANDED=0: GOTO 870
830       IF ASC(H$)=127 AND REVERSE=0 THEN
   REVERSE=1: PRINT #2,"<R>": PRINT "<R>":
   GOTO 870
840       IF ASC(H$)=127 AND REVERSE=1 THEN
   REVERSE=0: EXPANDED=0: GOTO 870
850       IF ASC(H$)>127 AND EXPANDED=0 THEN
   PRINT #2,CHR$(ASC(H$)-128);" ";: PRINT
   CHR$(ASC(H$)-128);" ";: EXPANDED=1: GOTO
   870
860       IF ASC(H$)>127 AND EXPANDED=1 THEN
   PRINT #2,SPACE$(ASC(H$)-128);: PRINT
   SPACE$(ASC(H$)-128);
870       GOSUB 920
880    WEND
890    CLS
900 NEXT I
910 PRINT #2,: PRINT #2,"<E>": CLOSE:
   SYSTEM
920 REM - GET A CHARACTER FROM HELP FILE
940 GET 1,RECNO
950 RETURN
960 REM - ERROR HANDLING
970 IF ERR=53 AND ERL=390 THEN RESUME 410:
   REM - Input file not found
980 IF ERR=53 AND ERL=450 THEN RESUME 470:
   REM - Output file not found
990 PRINT ERRS$+"-line "+STR$(ERL): STOP
1000 RESUME



10 REM ------ TXT2HLP/BAS
20 CLS: PRINT CHR$(16);SPACE$(35);"TXT2HLP/
   BAS";SPACE$(34);CHR$(17);
30 PRINT SPACE$(21);"Convert a TEXT file to
   DOS HELP format";SPACE$(21);:PRINT
40 PRINT SPACE$(14);
   "-------------------------------"
50 PRINT SPACE$(14);"|     Authored By
   Brian Davis, July 19, 1988       |"
60 PRINT SPACE$(14);"|1528 Wiggins Ave.|"
70 PRINT SPACE$(14);"|Saskatoon, Sask  |"
80 PRINT SPACE$(14);"|Canada S7H 2J8   |"
90 PRINT SPACE$(14);"|                 |"
100 PRINT SPACE$(14);"| This program car-
   ries no warranty of any kind and |"
110 PRINT SPACE$(14);"| the author assumes
   no liability for direct or    |"
120 PRINT SPACE$(14);"| incidental damages
   as a result of its use. It    |"
130 PRINT SPACE$(14);"| may be copied and
   distributed freely as long as  |"
140 PRINT SPACE$(14);"| the author identi-
   fication remains intact.       |"
150 PRINT SPACE$(14);
   "-------------------------------"
160 PRINT
170 REM - VARIABLE INITIALIZATION
180 TXTFILE$="": REM  name of input text file
190 HLPFILE$="": REM  name of output file
   in DOS/HLP format
200 H$="": REM - buffer string for output
   to random access help file
```

```
210 RECNO=0: REM  record number of ouput file
220 OUTCHAR$="": REM - 1 character of
   converted text to output
230 BUF4$="": REM - A 4 character input
   buffer for character examination
240 OUTLINE$="":REM  directory entry to output
250 YN$="": REM Storage for yes/no response
260 DIRNUM=0: REM - directory entry number
270 I=0: REM - general counter
280 J=0: REM - general counter
290 K=0: REM - general counter
300 DIM DIRTXT$(260): REM - array of direc-
   tory name entries starting at 0
310   REM - enough entries for 3 directory
   display pages
320   REM - of 76 entries each
330 DIM DIRADD(260,2): REM - array of high/
   low directory address bytes
340 HI=0: REM - array location for high
   order byte (for program readability)
350 LOW=1: REM - array location of low
   order address byte
360 DIRSTART=0: REM - the starting address
   of the directory section.
370  REM - This becomes the last two bytes
   in the output file.
380 COMPRESS=0: REM - A flag to indicate
   that compression occured on a space
390  REM - after a word. If a single char-
   acter has a space before
400  REM - and after it; the character and
   following space are
410  REM - not compressed
420 CHARCOUNT=3: REM - Count of characters
   output to file. The 3
430 REM - offset accounts for the 2 direc-
   tory address bytes at
440 REM - the end of the file and 1 HEX 0C
450 REM - item separator output before the
   directory.
460 REM -GET FILENAMES
470 REM - The input filename is a standard
   text file and has a logical
480 REM - record length of 256. The output
   filename is a random access
490 REM - file and has a LRL of 1.
500 ON ERROR GOTO 1600
510 INPUT "Enter INPUT TEXT FILENAME
   (prgname/ext.password:d) - ",TXTFILE$
520 OPEN "I",1,TXTFILE$: GOTO 550: REM - If
   this fails, the RESUME will
530    REM - print the error
540 PRINT "File not found!": GOTO 510
550 INPUT "Enter OUTPUT HELP FILENAME
   (prgname/HLP.password:d) - ",HLPFILE$
560 IF INSTR(HLPFILE$,"/HLP")=0 AND
   INSTR(HLPFILE$,"/hlp")=0 THEN PRINT:
   PRINT "The output HELP filename must
   have a '/HLP' extension": GOTO 550
570 OPEN "I",2,HLPFILE$: CLOSE 2: INPUT
   "File exists!...Overwrite?(Y or N) -
   ",YN$: IF YN$="N" OR YN$="n" THEN PRINT:
   GOTO 550 ELSE KILL HLPFILE$
580  REM - If the file does not exist, the
   RESUME will continue
590 ON ERROR GOTO 0
600 OPEN "R",2,HLPFILE$,1
610 FIELD 2,1 AS H$
620 REM -GET CHARACTER FROM INPUT FILE,
   TEST, AND REFORMAT
630 GOSUB 780: REM - get characters and
   fill buffer
640 WHILE NOT EOF(1) AND
   BUF4$<>"<E>"+CHR$(13)
650    IF BUF4$="<D>"+CHR$(13) THEN IF
   CHARCOUNT > 65535!-21 THEN PRINT: PRINT
   "*** OUTPUT CHARACTER COUNT WILL EXCEED
   65535 LIMIT ***": CHARCOUNT=0: INPUT
   "PRESS <ENTER> TO CREATE HELP FILE WITH
   ITEMS ENCOUNTERED SO FAR...",YN$: GOSUB
   1360: GOTO 770
660  REM - The "21" represents the max
   directory entry length (19)
670 REM  and the 2 directory address bytes.
680    IF BUF4$="<D>"+CHR$(13) THEN IF
   DIRNUM=260 THEN PRINT: PRINT "*** TOO
   MANY DIRECTORY ENTRIES IN SOURCE FILE
   ***": INPUT "PRESS <ENTER> TO CREATE
   HELP FILE WITH ITEMS ENCOUNTERED SO
   FAR...",YN$: GOSUB 1360:GOTO 770 ELSE
   GOSUB 1010 GOTO 750
690    IF BUF4$="<R>"+CHR$(13) THEN GOSUB
   1190: GOTO 750
700    IF LEFT$(BUF4$,1)<>CHR$(32) AND
   MID$(BUF4$,2,1)=CHR$(32) AND COMPRESS=0
   THEN GOSUB 1510: GOTO 740
710    IF LEFT$(BUF4$,1)<>CHR$(32) AND
   MID$(BUF4$,2,1)=CHR$(32) AND COMPRESS=1
   THEN GOSUB 840: GOSUB 780: GOSUB 870:
   GOTO 740: REM - output 2 char
720  REM - without compression
730    IF LEFT$(BUF4$,1)=CHR$(32) AND COM-
   PRESS=1 THEN GOSUB 1550
740    GOSUB 840: GOSUB 780: GOSUB 870
750 WEND
760 IF EOF(1) AND BUF4$<>"<E>"+CHR$(13)
   THEN PRINT: PRINT "*** END OF INPUT FILE
   ENCOUNTERED ***": INPUT "PRESS <ENTER>
   TO CREATE HELP FILE WITH ITEMS ENCOUN-
   TERED SO FAR..",YN$
770 GOSUB 1360: CLOSE: SYSTEM
780 REM GET INPUT CHARS INTO RIGHT OF BUFFER
790 WHILE NOT EOF(1) AND LEN(BUF4$)<4
800    BUF4$=BUF4$+INPUT$(1,1): REM - The
   buffer is filled from the right
810    PRINT RIGHT$(BUF4$,1);
820 WEND
830 RETURN
840 REM REMOVE A CHARACTER FROM LEFT OF BUFFER
850 OUTCHAR$=LEFT$(BUF4$,1):
   BUF4$=RIGHT$(BUF4$,3)
860 RETURN
870 REM OUTPUT A CHARACTER TO THE HELP FILE
880 IF ASC(OUTCHAR$)>127 THEN COMPRESS=1
   ELSE COMPRESS=0
890 RECNO=RECNO+1
900 CHARCOUNT=CHARCOUNT+1
910 IF CHARCOUNT=65535! THEN PRINT: PRINT:
```

```
   PRINT "*** OUTPUT CHARACTER COUNT EX-
   CEEDS 65535 LIMIT ***": CHARCOUNT=0:
   INPUT "PRESS <ENTER> TO CREATE HELP FILE
   WITH ITEMS ENCOUNTERED SO FAR...",YN$:
   GOSUB 1360: GOTO 770
 920 LSET H$=OUTCHAR$
 930 PUT 2,RECNO
 940 RETURN
 950 REM  OUTPUT A DIRECTORY ENTRY TO THE FILE
 960 FOR K=1 TO LEN(OUTLINE$)
 970   OUTCHAR$=MID$(OUTLINE$,K,1)
 980   GOSUB 870
 990 NEXT K
1000 RETURN
1010 REM - ADD TO DIRECTORY ARRAY
1020 IF RECNO>1 THEN OUTCHAR$=CHR$(12):
   GOSUB 870: REM - Output item separator
1030 FOR I=1 TO 5: GOSUB 840: GOSUB 780:
   NEXT I: REM - Get past CR at end of
1040  REM - <D> line and get next char.
   OUTCHAR$ is first char of
1050  REM - directory line.
1060 DIRNUM=DIRNUM+1
1070 WHILE OUTCHAR$<>CHR$(13): REM - Check
   for end of line
1080   DIRTXT$(DIRNUM)=
   DIRTXT$(DIRNUM)+OUTCHAR$
1090   GOSUB 840: GOSUB 780
1100 WEND
1110 IF LEN(DIRTXT$(DIRNUM))>19 THEN
   DIRTXT$(DIRNUM) =
   LEFT$(DIRTXT$(DIRNUM),19): PRINT: PRINT
   "DIRECTORY ITEM TITLE TRUNCATED TO
   "+DIRTXT$(DIRNUM): INPUT "PRESS <ENTER>
   TO CONTINUE...",YN$
1120 DIRTXT$(DIRNUM)=LEFT$(DIRTXT$(DIRNUM),
   LEN(DIRTXT$(DIRNUM))-1)+
   CHR$(ASC(RIGHT$(DIRTXT$(DIRNUM),1))+128)
1130 DIRADD(DIRNUM,LOW)=RECNO-(INT(RECNO/
   256)*256): REM - RECNO points to
1140  REM - first character of text de-
   scription for directory entry.
1150 DIRADD(DIRNUM,HI)=INT(RECNO/256)
1160 CHARCOUNT=CHARCOUNT +
   LEN(DIRTXT$(DIRNUM)) + 2: REM - The "2"
   represents
1170   REM - 2 address bytes for each
   directory entry
1180 RETURN
1190 REM - MAKE LINE REVERSE VIDEO
1200 OUTCHAR$=CHR$(127): GOSUB 870
1210 FOR I=1 TO 4: GOSUB 840: GOSUB 780:
   NEXT I: REM - Get past CR at end
1220  REM -  of <R> line and get next char
1230 IF LEFT$(BUF4$,2)=SPACE$(2) THEN GOSUB
   840: GOSUB 780: GOSUB 870: GOSUB 1550:
   GOSUB 840: GOSUB 780: GOSUB 870: REM -
   space compression codes cannot
1240  REM - follow reverse video codes. If
   the reverse video line
1250 REM - begins with 1 or more spaces,
   output 1 space and compress the rest.
1270 WHILE LEFT$(BUF4$,1)<>CHR$(13): REM -
   Check for end of line
1280   IF LEFT$(BUF4$,1)<>CHR$(32) AND
   MID$(BUF4$,2,1)=CHR$(32) AND COMPRESS=0
   THEN GOSUB 1510: GOTO 1320
1290   IF LEFT$(BUF4$,1)<>CHR$(32) AND
   MID$(BUF4$,2,1)=CHR$(32) AND COMPRESS=1
   THEN GOSUB 840: GOSUB 780: GOSUB 870:
   GOTO 1320: REM - output 2 char
1300 REM - without compression
1310   IF LEFT$(BUF4$,1)=CHR$(32) AND
   COMPRESS=1 THEN GOSUB 1550
1320   GOSUB 840: GOSUB 780: GOSUB 870
1330 WEND
1340 OUTCHAR$=CHR$(127): GOSUB 870
1350 RETURN
1360 REM - OUTPUT DIRECTORY TO FILE
1370 CLS: PRINT "OUTPUTTING DIRECTORY
   ENTRIES TO FILE..."
1380 OUTCHAR$=CHR$(12): GOSUB 870
1390 DIRSTART=RECNO
1400 FOR I=1 TO DIRNUM
1410   OUTLINE$=DIRTXT$(I)+
   CHR$(DIRADD(I,LOW))+ CHR$(DIRADD(I,HI))
1420   PRINT "ADDRESS = ";
1430   IF LEN(HEX$(DIRADD(I,HI)))<2 THEN
   PRINT "0";HEX$(DIRADD(I,HI)); ELSE PRINT
   HEX$(DIRADD(I,HI));
1440   IF LEN(HEX$(DIRADD(I,LOW)))<2 THEN
   PRINT "0";HEX$(DIRADD(I,LOW)); ELSE
   PRINT HEX$(DIRADD(I,LOW));
1450   PRINT "  ENTRY = ";
   LEFT$(DIRTXT$(I),LEN(DIRTXT$(I))-1);
   CHR$(ASC(RIGHT$(DIRTXT$(I),1))-128)
1460   GOSUB 950
1470 NEXT I
1480 OUTLINE$=CHR$(DIRSTART - (INT (DIRSTART
   /256)*256))+CHR$INT(DIRSTART/256))
1490 GOSUB 950
1500 RETURN
1510 REM  COMPRESS A SPACE FOLLOWING A CHAR
1520 BUF4$=CHR$(ASC(LEFT$(BUF4$,1))+128) +
   RIGHT$(BUF4$,2): REM - Compress
1530 GOSUB 780: REM - Get char to replace
   compressed space
1540 RETURN
1550 REM - COMPRESS MULTIPLE SPACES
1560 J=1
1570 IF MID$(BUF4$,2,1)=CHR$(32) THEN
   J=J+1: GOSUB 840: GOSUB 780: GOTO 1570
1580 BUF4$=CHR$(128+J)+RIGHT$(BUF4$,3)
1590 RETURN
1600 REM - ERROR HANDLING
1610 IF ERR=53 AND ERL=520 THEN RESUME 540:
   REM - input file not found
1620 IF ERR=53 AND ERL=570 THEN RESUME 590:
   REM - output file not found
1630 PRINT ERRS$ "IN LINE" ERL: STOP
1640 RESUME
```

(2) The line following a <D> line is the directory name associated with the text that follows. It is used as the search key to find the text in the file and display it. It is usually a program command but may be any text except the text format flags "<D>", "<R>", and "<E>". In the DOS help file, the standard is for directory names to be in UPPER CASE only. The maximum length of a directory entry is 19 characters since directory display columns are spaced every 20 characters.

(3) To highlight command syntax (or anything else you want) in reverse video place "<R>" on a line by itself and put the text to be highlighted on the next line.

(4) The last line of the source file must have "<E>" on a line by itself. It signals the end of the source file and that the directory section should be created and output. Using this method of file termination makes it easier to test for the end-of-file in the 4 character testing buffer. It also allows you to artificially generate and end-of-file condition at some other point for testing purposes.

(5) End each line of text with a carriage return (including the <D>, <R>, and <E> lines.

(6) The DOS HELP program will display up to 24 lines of text and wait for any key to be pressed (it does not prompt for a key to be pressed). On the first page of the display for a help item, the directory name is displayed as the first line leaving 23 lines for text. To make the display more user friendly, place a prompt such as "Press any key for more" at the last line that will be displayed. If you want to break the text at an earlier point in the help item text and still continue on the next page, fill the empty space with blank lines (carriage returns in the file).

(7) The source file entries (and ultimately the directory entries) must be in alphanumeric order of directory name so the DOS HELP program can find them. All directory entries will be shown in the directory display, but if the entries are out-of-order, the out-of-order entries will be inaccessible since the DOS Help/cmd program will stop looking for the requested item as soon as it encounters a directory entry with a higher "sort value" than the item requested.

(8) Reverse video sections that were not on a separate line in the source help file (such as in some sections of the original DOS/HLP file) will be formatted incorrectly in the text file resulting from the conversion. The added complexity of detecting an inline reverse video flag wasn't considered to be worth the trouble. The current method allows entry of any text provided that the text format flags appear on lines by themselves and that the flags are not unintentionally entered as text on lines by themselves (ie. the flags can be entered anywhere else within a line as shown above).

(9) The maximum number of characters in the source file is approximately 65535 since the HELP file has a record length of 1 and the maximum number of records that can be addressed in a file is 65535. However, the compression of the HELP version of the file will actually allow more characters to be entered. The program checks the output file size as it is created. If the limit (including directory entries) is exceeded, it stops processing the source and completes the output file with the entries processed up to that point.

DOS HELP FORMAT

The output help file format follows:

(1) The logical record length of the file is 1.

(2) Each help item/page is separated/identified by a form feed character (decimal 12, hex 0C).

(3) A single space following a non-space character is compressed by adding decimal 128 (hex 80) to the ASCII value of the character before the space and deleting the space. This also applies to carriage return characters followed by a space. (ie. if there are multiple spaces at the beginning of a line, the first space will be compressed on the previous carriage return and the rest of the spaces will be compressed by a space compression code. There is a slight difference for reverse video lines as described in the program description.)

(4) Multiple spaces are compressed using space compression codes. A decimal 129 (hex 81) generates 1 space, 130 (hex 82) generates 2 and so on. Multiple space codes do not include the first space after the previous non-space character. The first space is compressed as above.

(5) The start and end of a reverse video field is signalled with a decimal 127 (hex 7F) code.

(6) The start of the Directory section is signalled with a form feed as above. Each directory entry has its last character compressed as if it was followed by a space and is followed by the address of its corresponding text in the file; 2 bytes (low order, then high). However, spaces in the directory entries are not compressed.

(7) The last 2 bytes in the file indicate the address of the first entry in the directory; again 2 bytes (low order, then high).

(8) Lines of text are ended with a carriage return character (decimal 13, hex 0D. Blank lines are entered carriage returns.

(9) If a single character is preceded and followed by a space, the character and the following space are not compressed. If this occurred there would be 2 space compressions in a row and the second would be treated as a simple space compression by the HELP program, rather than a space compressed onto another character.

(10) The DOS help file text format consists of 4 sections. Section 1 is the name of the command you want help for. This is the first line of the display. Section 2 is the general description of the function of the command. Section 3 is the generalized syntax of the command in reverse video. Section 4 is the detailed description of the

command options. All keywords used in the command are in UPPER CASE.

(11) A directory may contain more than 76 entries but the DOS HELP program will only display 76 entries on the first page, 92 entries on each following page and no prompt is provided to tell the user that there is another page, or how to get to it. To see the entries on the next page the user must press the <ENTER> key at the keyword/break prompt. Pressing the <ENTER> key on the prompt for the last page cycles back to the first page. The help item to be displayed does not need to be shown on the screen when requested (ie. items on following directory pages can be requested on the first page). The programs presented here allow for 3 pages of directory entries (total 260).

The directory flag signals the end of the previous item and that the next line is a directory entry. Up to 76 directory entries may be displayed on the first page and up to 92 entries on following pages where each directory entry has a maximum of 19 characters per entry. The entries are displayed in 4 columns in the order entered in the file (left to right then top to bottom).

The reverse video command is used to highlight the actual syntax. Reverse video lines may be separated by text. Any number of lines of reverse video may be used. The text can be as long as it needs to be to describe all the details.

If more than 1 screenfull must be displayed, include a prompt to press any key and provide enough blank lines to fill 24 lines on the screen (the directory name is the first line and format flags are not counted). Additional text on the same line may be used to describe what is on the next page (eg. Press any key to see page 2 of 3...).

Program HLP2TXT/BAS

The first program "HLP2TXT/BAS" converts the DOS "Help" format file into text format which can be edited with any text editor. Note that the editor must be capable of producing a simple text output file without the formatting characters that some word processors generate and each line must end with a carriage return (Some word processors generate ASCII files with only 1 carriage return per paragraph).

LINES 10 TO 160: This section is the program identification and disclaimer.

LINES 170 TO 320: This section is the variable initialization. In addition to providing the initial contents of the variables used, I use this section as a summary list of the variables used and a description of the purpose of each. This documentation helps to debug or change the program in the future.

LINES 330 TO 490: These lines are used to get the input and output filenames. Error handling is turned on at the beginning of this section. After the input filename is entered, a test is made for its existence. If the filename entered does not exist, then the attempt to open the file at line 390 will generate an error which will cause the lines from 960 to 1000 to be evaluated. If the file does exist, then the program continues on to line 420 and opens the file in random access mode with a record length of 1 otherwise it resumes at line 410, displays the error and requests a new filename. The same sort of test is performed after the output file request. The output file specified is checked to see if it exists in order to prevent overwriting another file unintentionally.

LINES 500 TO 720: This area reads the directory section of the help file. There is no check made to see if the file is a valid help file, so if the wrong filename was entered either garbage will be produced or a fatal error will result.

Lines 530 to 550 determine the last record number in the file. BASIC is a little inconsistent in that the LOF function (which returns the last record number in the file) returns values from -32766 to +32767 while the PUT and GET functions require 1 to 65535 as record numbers. Therefore a conversion is made to make negative returns from the LOF function compatible with the GET and PUT functions. Lines 560 to 580 get the last 2 bytes of the help file to determine the starting record of the file directory. The loop from line 600 to 720 then moves through the directory and loads the directory titles into the DIRTXT$ array and the addresses of the beginning of the corresponding text blocks into the DIRADD array. The last character of each directory entry must be expanded from its compressed format.

LINES 730 TO 910: These lines reformat the text block entries and output the resulting text to the specified file. The form feed character (decimal 12, hex 0C) is used as the text block separator in the help file and this program uses it as a flag to end the conversion for a block. Reverse video entries in the source file are preceded and followed by the decimal 127 (hex 7F) character. These are stripped and replaced by a "<R>" sequence preceding the text that was in reverse video. Expansion of compressed spaces is also performed according to the rules explained above.

This is the subroutine to read each character from the source file.

Program TXT2HLP/BAS

In general the program reads the source file 1 character at a time into a 4 character buffer. As each character is read the buffer is checked for the the special character sequences "<D>", "<R>", or "<E>" (each followed by a carriage return). If the sequences are not detected then the characters are processed according to the space compression rules (described in the sample text file above). If the "<D>" sequence is detected then a directory entry is created and stored in an array (DIRTXT$ for the title and DIRADD for the address) to be output at the end of the file. If the "<R>" sequence is detected then the reverse video characters are output before and after the next line of characters. If the "<E>" sequence is detected then processing of text stops, the directory is output to the file and

the program ends. Note that each of these sequences must be on a line by themselves or they will be treated as normal text. While characters are being output, a running count of the output file length is kept (including directory entries) so that if the source file is too big, a help file is still created with the text processed to that point.

Note that in generating directory addresses (breaking the record number into 2 bytes), the MOD function was not used since it requires arguments in the range of -32766 to +32767 and record number ranges from 1 to 65535.

LINES 10 TO 610: These lines are the program description, disclaimer, variable initialization, and filename input sections as described above. One difference is that when a file is opened in random mode, its contents are not erased before writing to it so to prevent previously written information from overlapping the new, the file is first deleted and then opened.

LINES 620 TO 770: These lines make up the main loop of the program which calls the other subroutines to do the processing. The first IF statement in the loop checks to see if the running character count has been exceeded. A limit is set to 21 characters less than the maximum to account for the possibility that the character count may be exceeded when the next directory entry is read (since the character count is updated with the entire length of the directory entry (line 1160) rather than 1 character at a time as for regular text (line 900)).

The second IF statement checks to see if the maximum number of directory entries has been exceeded. If the directory entry or character count limit has been exceeded then the user is prompted to produce a help file with the information processed to that point.

If the limits were not exceeded and the "<D>" sequence was detected, then the subroutine at lines 1010 to 1180 is called. An end of block form feed character is output to the file, the 4 character buffer is cleared and filled with new characters, the

```
10 REM ------ TESTHLP/BAS
20 CLS: PRINT CHR$(16);SPACE$(35);"TESTHLP/
   BAS";SPACE$(34);CHR$(17);
30 PRINT SPACE$(20);"Generate a test text file for TXT2HLP/
   BAS";SPACE$(19);:PRINT
40 PRINT SPACE$(14);
   "-----------------------------------"
50 PRINT SPACE$(14);"|     Authored By Brian Davis, July
   19, 1988       |"
60 PRINT SPACE$(14);"|                    1528 Wiggins Ave. |"
70 PRINT SPACE$(14);"|                    Saskatoon, Sask   |"
80 PRINT SPACE$(14);"|                    Canada S7H 2J8    |"
90 PRINT SPACE$(14);"|   CAUTION: THIS PROGRAM HAS NO ERROR
   CHECKING     |"
100 PRINT SPACE$(14);"| This program carries no warranty of
   any kind and |"
110 PRINT SPACE$(14);"| the author assumes no liability for
   direct or    |"
120 PRINT SPACE$(14);"| incidental damages as a result of
   its use. It  |"
130 PRINT SPACE$(14);"may be copied and distributed freely
   as long as   |"
140 PRINT SPACE$(14);"the author identification remains intact "
150 PRINT SPACE$(14);
   "-----------------------------------"
160 PRINT
170 INPUT "Enter output filename - ",FILE$
180 INPUT "Enter the number of directory entries (1-260) -
   ",ENTRIES
190 INPUT "Enter the number of text lines per entry - ",LINES
200 INPUT "Enter the number of characters per text line - ",CHAR
210 OPEN "O",1,FILE$
220 FOR I=1 TO ENTRIES
230 PRINT #1,"<D>"
240 PRINT #1, "ITEM";: PRINT #1, USING "###";I:PRINT
   "ITEM";: PRINT USING "###";I
250 PRINT #1,"THIS IS ITEM #"+STR$(I)
260 PRINT #1,"<R>"
270 PRINT #1,"I SAID, THIS IS ITEM #"+STR$(I)
280 FOR J=1 TO LINES: PRINT #1,STRING$(CHAR,"H"):NEXT J
290 NEXT I
300 PRINT #1,"<E>"
310 CLOSE 1
```

directory text entry is added to the DIRTXT$ array and the address of the corresponding text block is added to the DIRADD array.

If the "<R>" sequence was detected then the next line is output preceded and followed by the reverse video toggle character using the subroutine at line 1190.

If the "<E>" sequence was detected then processing of text is terminated, the directory is output, the files are closed, and the program exits to the DOS prompt.

Lines 700 to 730 implement space compression according to the following rules: (consider a sequence of 4 characters where the first character has been processed and output and the others are the first three characters in the 4 character buffer)

1) if the second character is not a space, the third character (the second in the buffer) is a space and the first character was not compressed then compress the third character (the space) onto the second character.

2) if the second character is not a space, the third character is a space and the first character was compressed then output the second and third characters without compression. This case applies to a single non-

space character surrounded by spaces. It prevents outputting two compressed characters in a row which would be translated by the HELP/CMD program as a space compressed onto a character followed by a space compression code. 3) if the first character was compressed and the second character (the first in the buffer) is a space then convert the second character and any following spaces into a space compression code.

Line 740 outputs a character to the output file.

LINES 780 TO 830: This subroutine gets characters from the source file and brings them into the right side of the buffer until it is full. (Certain subroutines remove more than 1 character from the buffer at a time.) It also echoes the input text to the screen.

LINES 840 TO 860: This subroutine takes one character from the left side of the buffer and places it into a holding string variable for output.

LINES 870 TO 940: This subroutine outputs a character to the output file and updates the character counter.

LINES 950 TO 1000: This subroutine outputs a directory entry to the output file.

LINES 1010 TO 1180: This subroutine adds a directory entry to the array. The directory text line is stored in DIRTXT$ and the address of its corresponding text block is stored in DIRADD. The character counter is updated in this section so that there will be enough room for the directory in the output file if there happens to be too much text in the source file.

LINES 1190 to 1350: This subroutine outputs a line with the reverse video toggle characters before and after it. It first outputs a toggle character and then it throws away the the "<R>" in the buffer as well as the following carriage return. It then outputs the text line with space compression. Space compression codes are ignored by the HELP/CMD program if they follow the reverse video toggle code so if the reverse video text line starts with spaces they are output as follows:

1 One to two spaces - output without compression.

2 Three spaces - first space output without compression, the third space compressed onto the second.

3 More than three spaces - same as #2 with the additional spaces converted to a space compression code.

The reverse video toggles are output after the carriage return on the previous line and before the carriage return on the reverse video line to restrict reverse video to only the one line.

LINES 1360 TO 1500: This subroutine outputs the stored directory entries to the file when regular text processing is terminated because the "<E>" sequence was detected, the end of the source file was reached, or the maximum character count was reached. The directory entries and addresses are echoed to the screen as they are output.

LINES 1510 TO 1540: This subroutine compresses a single space following a character by throwing away the space and adding decimal 128 (hex 80) to the character.

LINES 1550 TO 1590: This subroutine compresses multiple spaces by converting them to space compression codes. (The first space in the string is output as is, or is compressed onto the previous character before this subroutine is called.)

LINES 1600 TO 1640: This error handling routine works as described for the HLP2TXT/BAS program above.

Program TESTHLP/BAS Description:

This is a simple program to generate properly formatted text files of selected sizes to test the TXT2HLP/BAS program. It does not contain error checking so be careful with your choice of file names. The operation of the program needs no description.

CONCLUSION:

The DOS HELP/CMD program is particularly useful when Double Duty program is used since you can interrupt whatever you are doing and get help. With these programs and a simple text editor, you can create help files for use on the Model 4 that give you compact and portable storage/backup for your documentation. All that remains to be done is the typing.

Let our LB Data Manager solve your data storage problems

LB Version 2.2: A Flat File Data Manager with more powerful and easy to use features in this latest enhancement of Little Brother! Now with data conversion utility for DIF, pfs, Profile, dBASE, ...

We've added many features asked for over the past few years by LB users; yet LB is still about the easiest, most flexible data manager you can use for managing your data. Absolutely no programming is needed to create a database with numerous fields, construct input screens for adding and editing data, and create your own customized report. Quickly you define your data fields in response to LB's prompts, and then draw your data input screen using simple keystrokes - or have LB automatically create your input screen. In no time at all, you're entering data. Customize your printed reports with user-definable print screen definitions. LB is just what you need in a data manager! Now even more in version 2.2!

Data capacity per database:
LB supports up to 65,534 records per data base; 1,024 characters (64 fields) per record; and up to 254 characters per field.

Field types supported:
LB allows ten field types for flexibility: *alphabetic* {A-Z, a-z}, *calculated* {operations on "numeric" fields using +, -, *, /, with 2-level parentheses}, *date last modified* {YYYY/MM/DD automatically maintained}, *dollar* {±ddddddd.dd}, *floating point* {±ddddddd.ddddddd, *literal* {any ASCII character}, *numeric* {0-9, -, .}, *right-justified numeric*, *upper case alphabetic* {A-Z, automatic conversion of a-z}, and *upper case literal* { literal with automatic conversion of a-z}. All field types utilize input editing verification so invalid data cannot be added to a record. Field name strings can be up to 19 characters long.

Data entry and editing:
LB allows you to design up to ten different input/update screens to provide extreme flexibility for selectively viewing your database fields. Using a database password provides the capability of selectively protecting fields from being displayed or printed without entry of the correct database password, or they can be protected from being altered. This is quite useful in a work-group environment. Fields may selectively be established to require a data entry before a record being added or edited is saved. You can enable a special index file to keep track of records being added. This can be subsequently used, for example, for a special mailing to newly added *customers*. Flexible editing includes global search and replace with wild-card character match and source string substitution. Search and replace can be performed on all records, or on records referenced in an unsorted or sorted index file.

Record selection and sorting:
You can maintain up to ten different index files to keep your data organized per your multiple specifications. Records may be selected for reference in an index file by search criteria using six different field comparisons: EQ, NE, GT, GE, LT, and LE. You can select on up to eight different fields with AND and OR connectives. Index files can be left unsorted, or you can sort in ascending or descending order. By attaching a sorted index file, any record may be found within seconds - even in a very large database. **LB even includes a special command for automatically finding duplicate records!**

Report generation:
Report generation incorporates a great flexibility. Your report presentation is totally customized through print definition formats which you define on the screen as easily as you define the input/update screens. You can truncate field data, strip trailing spaces, or tab to a column. You control exactly where you want each field to appear. LB provides for a report header complete with database statistics: database name, date, time, and page numbers. A report footer provides subtotaling, totaling, and averaging for dollar, floating point, and calculated fields; print number of records printed per page and per report. Attach any of the ten index files and you control exactly what records get printed; even a subset of indexed records can be selected for printing to give you a means of recovering from that printer jam halfway through your 30-page printout. You can even force a new page when the key field of an index file changes value. Up to ten different printout definition formats can be maintained for each database. Reports may be sent easily to a printer, the console display screen, or to a disk file - useful for subsequent printing or downstream data export to other programs. Report formatting allows for multiple across mailing labels, multiple copies of the same record, or even printing one record per page for sales books. You can easily generate mail/merge files of address or other data for your word processor. Or you can use LB's built-in form letter capability.

Automatic operation:
For automating your processing needs, LB can be run in an *automatic* mode, without operator intervention. Frequently used procedures can be saved by LB's built-in macro recorder for future use. Entire job streams may be produced, so that LB operations may be intermixed with literally any DOS function that can be *batch* processed.

Maintenance utilities:
To make it easy for you to grow your database as your data needs grow, we provide two utility programs for managing your database. One allows you to construct a new database with an altered data structure and populate it with data from your existing database Another allows you to duplicate your database structure, copy or move records from one to another, or automatically purge un-needed records. **A third utility converts to LB from pfsFILE4, Profile4, DIF, dBASE II&III, and fixed record; also converts to DIF, dBASE, and delimited.**

Help is on the way:
The main menu even provides a shell to DOS so you can temporarily exit LB to perform other DOS commands. LB provides extensive on-line help available from almost every sub-command. A 200-page User Manual documents every facet of LB's operation.

Trade-up policy:
Send in an original Table of Contents page from any existing database program and get LB Version 2 for half price.

Specify MS-DOS or TRS-80 version. LB is priced at $99 + $5 S&H (US; $6 Canada; $7 Europe; $9 Asia, Pacific Rim, and Australia). To trade up from any other database, send Table of Contents page and $49.50 +S&H. Remit to:

MISOSYS, Inc.
PO Box 239
Sterling, VA 20167-0239
703-450-4181 or orders to 800-MISOSYS

MISOSYS, Inc.

MISOSYS sponsors a forum on CompuServe: PCS49

When you don't have to write in stone, don't let your editor weigh you down. You need SAID-86!

Editing was never so easy!

SAID-86 is a fast, flexible, full screen text editor for PC's. It is perfect for editing batch files, program listings, README files, CONFIG.SYS files, and anything you now do with EDLIN or the non-document mode of a word processor. Why struggle with huge editors; when all is said and done, SAID-86 will be your text editor of choice!

Check out this list of features

- ✓ WordStar-like editing commands are easy to use
- ✓ Pull-down menu system for commanding SAID-86
- ✓ Supports nine editing buffers with automatic swap to disk
- ✓ Supports up to 30 user-defined macros; 255 characters each
- ✓ Undelete the last nine deleted lines can save your bacon
- ✓ MOUSE support with automatic recognition
- ✓ HELP facility; shell to invoke DOS commands from SAID-86
- ✓ SAID-86 can expand or contract TABs

SAID-86 is reasonably priced at just $29.95 + $3S&H

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 20167-0239
800-MISOSYS
(US&Canada)
or 703-450-4181
M/C & VISA accepted.
S&H are U.S. only.

TRSCROSS

Now you can transfer TRS-80 Model III/4 files directly to your MS-DOS disks right on your PC. Convert BASIC programs; Convert SuperScripsit document files to DCA-RFT. Only $89.95 + $4S&H

HartFORTH-86™

HartFORTH is a Direct Threaded Code implementation full 79-STANDARD FORTH which runs under DOS; the Virtual Memory that it accesses for storage and retrieval purposes is a file created and controlled by the operating system. HartFORTH's enhancements include functions to call the DOS file handling routines so that other files may be created and accessed if required. A library of standard screens is supplied with HartFORTH to provide often used extensions to the language, such as double length and floating point math, editing of source screens, string manipulation, arrays, etc. Priced at $59.95 + $5S&H

- HartFORTH programs can invoke other programs via **EXEC** and **EXEC.PROG**.
- Functions create new files from within HartFORTH, and allow the current Virtual Memory file to be changed for another and manipulated at the individual block level.
- Provides the recommended 79-STANDARD DOUBLE NUMBER STANDARD EXTENSION word set that implements 32-bit operations.
- **CASE:** and **SWITCH:** functions allow multi-way branching decisions to be taken with execution continuing in-line once the word branched to completes.
- String manipulators include: `"VARIABLE`, `"CONSTANT`, `"!`, `"LEFT`, `"RIGHT`, `"MID`, `"+`, `"COMPARE`, and `">`.
- DOS software and hardware interrupt vector access support via: `GET.VECTOR`, `PUT.VECTOR`, `THIS.SEG`, `DI`, and `EI`.
- V24 program input, output, and interrupt input support.
- *Overlay management words*: `FORGET.OVLY`, `OVLYNAME`, `PUT.DATA`, `OVLY.ENTRY`, `SAVE.OVLY`, `CORRECT?`, `LOAD.OVLY`, `NEW.OVLY`, `RUN.OVLY`, and `LEAVE.OVLY`.
- Screens provide trigonometric functions: `SIN.`, `COS.`, `TAN.`, `SIN`, `COS`, and `TAN`.

Are you still fussing with floppies for BACKUP? CMS' DJ10 or DJ20 tape drive from MISOSYS is your solution!

The Colorado Memory Systems' JUMBO tape drives fit all computers. Internal mounting in AT's, XT's, and PC's, they connect to your floppy disk controller. Tape adaptor board needed when two floppies are in use. Kit converts Jumbo to external use.

- In about 5.5 minutes, a DJ10 backs up 10MB's file-by-file - the fastest in the industry! 40MB's gets backed up in about 18 to 20 minutes. Uses industry-standard DC2000/DC2120 tape cartridges.
- DJ10/DJ20 plugs into your floppy disk controller to save cost, power, and a slot. Needs 5-1/4" (or 3.5" with faceplate) mounting slot.
- Optional adapter board mounts in your computer to provide a tape port. When used with the external DJ10/DJ20, it lets you share your drive between computers. Note: external adaptor includes "Tape Adaptor"
- DJ10 has up to 120 megabytes of capacity using compression with a DC2120 tape; DJ20 has up to 250 megabytes of capacity.

DJ10 Jumbo	$199 ($7S&H)
DJ20 Tape drive	$265 ($7S&H)
A11 Adaptor	$45 ($3S&H)
K10 External Kit	$110 ($5S&H)
DC2000 tape	$20.00
DC2120 tape	$25.00

NEW LOWER PRICES

Why buy just a FAX board, when the ZOFAX 96/24 from MISOSYS includes a 2400 baud modem for a few bucks more? Turn your PC into a FAX machine!

- ✓ Send and receive FAX from any CCITT Group III Fax Machine or PC Fax
- ✓ Auto receive and print incoming Fax messages ✓ Background receiving
- ✓ Distribut Fax messages to multiple destinations ✓ Fax mail merge
- ✓ Time schedule transmission to take advantage of low nightime rates
- ✓ 2400 bps Fully Hayes Compatible Modem
- ✓ Includes powerful but easy to use BITCOM and BITFAX software

Further price reduction: Just $125 + $7 S&H

EXPANZ!™ Disk Expander Card

With the EXPANZ! data compression card, you can boost hard disk capacity up to three times. EXPANZ! plugs into any open slot and intercepts calls to and from the disk controllers. Compresses and decompresses in real time. Requires PC/XT/AT or compatible running DOS 3.x or higher. Now Just $150 + $7S&H.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 20167-0239

800-MISOSYS
or 703-450-4181

MISOSYS, Inc.

With a 20 or 40 MB MISOSYS Hard Drive connected to your TRS-80 Model III or 4, your computer will sail through data access.

Order any hard drive kit or unit from MISOSYS and we'll pre-install either LS-DOS 6.3.1 or LDOS 5.3.1 at no extra

MISOSYS has been shipping complete drive kit packages since September 1989 which plug into Model 4/4P/4D and Model III computers; let us build one up for you! Our host adaptor, which interfaces the 50-pin expansion port of the TRS-80 (host) to the 50-pin SCSI port of the HDC, sports a hardware real time clock using a DS1287 clock module. With its internal battery lifetime in excess of 10 years, never enter date and time again. It even adjusts for daylight saving time! An available option is a joystick port and Kraft MAZEMASTER joystick with a port interface identical to the old Alpha Products joystick; thus, any software which operated from that joystick will operate from this one.

Software supporting the S1421 and 4010A controllers includes: a low level formatter; an installation utility and driver; a high level formatter; a sub-disk partitioning utility; utilities to archive/restore the hard disk files onto/from floppy diskettes; a utility to park the drive's read/write head; a utility to set or read the hardware clock; a keyboard filter which allows the optional joystick to generate five keycodes; and a utility to change the joystick filter's generated "keystroke" values after installation. Optional LDOS 5.3 software is available.

Twenty megabyte drive packages are currently built with a Seagate ST225 hard drive; Forty megabyte packages use a Seagate ST251-1 28 millisecond drive. Drive packages are offered as '*pre-assembled kits*'. Your 'kit' will be assembled to order and fully tested; all you will need to do is plug it in and install the software. Drive kits include a 50-pin host interface cable and the hardware clock. Full implement of status lights included: power, ready, select, read, and write. Add a joystick for but $20 additional (see price schedule).

Aerocomp Hard Drives now available from MISOSYS

MISOSYS is also the sole source of remaining brand new Aerocomp hard drives. All Aerocomp drives include status LEDs, software driver and formatter, power and host cables, and installation Job Control Language. We are building their 20M and 40M drives. We also have Montezuma Micro CP/M Hard Disk Drive drivers available.

Prices currently in effect:

Complete MISOSYS Drive Kits:	
20 Megabyte kit:	$450
40 Megabyte kit:	$575
Joystick option	$20
LDOS software interface	$30
Aerocomp Hard Drives:	
20 Meg unit	$400
40 Meg drive	$500
MISOSYS H/A with software	$75
Separate Hard Disk Controllers	
Xebec 1421 HDC	$75
Adaptec 4010 HDC	$75
WD1002S-SHD	$75
Drive power Y cable	$5
XT drive cable set	$5

Note: freight charges are additional.
Prices subject to change without notice.

MISOSYS, Inc.
PO Box 239
Sterling, VA 20167-0239
U.S.A.

Contents: Printed Matter

BULK RATE
U. S. POSTAGE
PAID
Sterling, VA
PERMIT NO. 74

Attention Postmaster: Address Correction Requested
Forwarding and return postage guaranteed