Sept/Oct 1989

USUS NewsLetter

All the News that Fits, We Print

Volume 3

William D. Smith, Editor

Number 7

From the Editor

by William D. Smith

My apologies! This month the NewsLetter is late because I didn't get it done until October 6. I have just started a full time project. I haven't been working full time in the last 5 years and therefor had time to do things of interest to me like this NewsLetter.

I will be unable to do the NewsLetter after the first of the year (barely getting it out now). Anyone interested in taking up the job may contact me at (619) 941-4452 for more information. I will put out the next NewsLetter ("Nov./Dec. 89") and hopefully help someone put out the "Jan./Feb. 90" one.

Hays Busch will also be resigning at the end of the year. We need someone (or more then one person) to take over the work he is doing. Please contact Hays at (303) 526-0057 for more details on what he does.

Although we were planing to have the Board of Directors elections this NewsLetter, the Board postponed them since we only had two candidates. There will be a special mailing sometime in November. If you at all interested in running for the Board, contact Alex Kleider at (415) 327-6197.

For those who use MUSUS, it will be changing (maybe by the time you read this) to the "The Portable Programming Forum". If "GO MUSUS" quits working, use "GO PASCAL".

News from Millennium (August 8, 1989)

by Bill Bonham

Millennium Computer Corporation has acquired the rights to build and sell the computer products previously produced by MicroSage Computer Systems. Millennium has recently moved to a new location. Our new address is: Millennium Computer Corporation, 954 Spice Island Drive, Sparks, Nevada 89431 (702) 331-6000, Fax (702) 331-6088.

Millennium has also begun shipping its own new product called the SuperStride 740. It is a 25MHz 68030 based system available in ether a vertical (tower) or horizonal (desktop) configuration. The base system is a single board computer with 16 serial ports, up to 16 megabytes of RAM, a SCSI Bus interface, and a VMEbus interface. RAM is expandable to 32 megabytes with a "Sky RAM Board" that piggybacks on the main CPU Board. Up to 32 additional serial ports may be added on two "Sky Port Boards" which also piggyback on the CPU Board. A "Sky Ethernet Board" with thin and thick cable Ethernet interfaces and 8 serial ports may be substituted in place of a Sky Port Board. Ethernet, however, is not supported under the Power System.

The vertical chassis is smaller than the Stride 460 and for better cooling and reliability, is not packed as full. Both the horizonal and vertical chassis will hold one Winchester drive (up to 700 megabytes unformatted), one 150 megabyte streaming tape drive, and one 5.25 or 3.5 inch floppy drive. The SCSI Bus and Floppy interface signals along with system power are made available for external expansion through connectors. Millennium will support External Enclosures to house additional drives.

Complete system hardware testing facilities are built into the resident PROMs. The drive formatting and qualification routines as well as Winchester partition management are in the PROMs. Also available in PROM is a version of the partition backup and restore facilities to streaming tape.

Millennium supports the Power System and UniStride (an implementation of UNIX) on the 740 product. The Power System runs under an expanded 31 user Multi-User BIOS. Improvements in configuration management, system installation, and access to system level functions have been made.

We Get Letters...

From W. Sheringo (#805)

re Vol. 3, No. 6

1. A package as proposed would be acceptable.
2. I don't know whether "not renewal's" will renew.
3. USUS w/ a journal??? — USUS w/o a journal??? USUS??? When one expends $$$, one expects to receive something of value in return. What are we getting for the buck? — more reading matter to see if there's something of value in return? — or, problems to divert our attention from what we are doing before we started to read it? Are we going to for business interests? — or are we going for hacker/pastime interests? — Is USUS another junk club trying to please a lot of people? — does USUS wish to attract intelligence/people-who-can-add-value/(fill in the blank _____)??? If we must, forget Pascal/Ada/Modula2. Problems must be formulated before they can be solved.

From Rick Osborn (#4070)

Mr. Spitzer,

In spite of the fact that my time, and soon my house, are swamped by journals, I am voting for the USUS membership bound with JPAM. The package as presented is acceptable to me. I don't know any ex-members to query about whether the package would draw them back.

Just another quick note regarding your Modula 2 String code in Vol 3, Issue 6 of USUS NewsLetter. I am some what puzzled by your 21 Feb 89 patch to 'copy'. I am assuming that this change was to handle strings of length zero. However, I cannot see any reason to handle zero as a special case. The 'IF HIGH' test should always fail. The 'FOR' loop should execute zero times, and the 'IF len' test should be equivalent to the special-case code included for len=zero. I wouldn't make such a big deal out of it except for the fact that the code looks right. If you had problems with the strings of length zero, rigorous code verification by inspection holds no hope for improving code quality. Either that, or the semantics provided by your compiler are very suspect.

Bob's reply

To Rick Osborn,

Actually, I thought I had coded it OK, but somehow one of the graduate students showed me a case where it failed and made the changes —there was a subtle case, maybe when the array had 0 length (HIGH = 0), but the procedure would check s[0] for a null and blow up with a range error. I forget the details. If I have time and can remember (both are becoming increasing rare) I will try to check this. I am pleased that the module is helpful (I hope), otherwise.

From William G. (Gordy) Kastner
TURQUOISE PRODUCTS
9648 E. Baker Street
Tucson, Arizona 85748
(602) 885-9671

USUS with a Journal JPAM would be great. I have subscribe to JPAM for several years. It would be a great magazine for USUS. I also subscribe to Computer Language and Dr. Dobbs and neither cover Pascal or Modula very well any more. I don't know any language magazine that does. They don't even mention that Modula2 is really object oriented. All language magazines are solid into C which is not near as good a language for programming large programs as Modula2.

$45 package for USUS membership and JPAM would be great.

I will renew in any event.

I am a Modula2 programmer and would like to see USUS into a lot more Modula2. I sell a large estimating program to printshops with ads in printing magazines. It is written in Logitech Modula2. I also have a shareware program called TURK TOOLS. It is just a simple little program that has simple procedures like how to write to a lineprinter, how to pick up the date from hardware, and how to program reverses and a lot more, most of which are not mentioned in most books on Modula2 including the Logitech manual. Sent a copy to USUS. Thank you.

MacWorld Trip
by Harry Baya and William Smith

My friend William Smith and I went to this big Macintosh thing in Boston a few weeks ago and we had a good time even though it rained a lot and not everything was perfect all the time.

William drove me up in his truck. He came about 50 miles out of his way to get me but he said he did not mind because when you are driving from California to Boston 50 miles does not seem like that much. We spent our one night in Boston at a fraternity house at MIT and that was sort of strange too but probably should be in a different newsletter.

Because we belong to this club called USUS this big company, Apple, invited us to breakfast one morning with lots of people from other clubs and they gave us a real nice meal with seconds and all only I am on a diet and William is thin so we did not eat all that much. I won a prize at this breakfast, as did about 100 other people. Mine was a Macintosh "Idea Processor" called "Inspiration" which was interesting because I had met the president of the company that made it earlier and she seemed pretty nice -only sort of busy. I have not used it yet but when I get some ideas that need processing I will try it.

William did not win anything at breakfast, but I got a business card that entitled me to a free copy of something that came in a nice carry bag so I gave that to William and he seemed to like it, only what was inside the bag turned out not be as interesting as we had hoped. I think it was a video tape saying how great something was or something like that.

I wore a homemade button that said USUS on it and Apple gave us tags that also said USUS and almost no-one had ever heard of us. Even when I told them who we were they did not get interested, usually. I wore one of my buttons on my Australian hat and maybe they were not ready for such wild eccentricity — though not everyone looked that normal there anyway.

We went to the exhibits, which were in two buildings, miles apart and each bigger than a football field and I never saw so many little booths with such enthusiastic people. My favorite one was a thing called "Cause" which was sort of a cross between a database system and a program making environment. It had a graphic interface that allowed the user to build the input and output screens by "dragging" things around the screen and the man who showed it to me made it look very easy. Only he was the president of the company and I think he created this thing so maybe it was a lot easier for him than it would be for me and, besides, I'm not sure about this "drag" stuff as people who are into that sort of thing are sometimes a little too weird.

We met one USUS professional member, **Eric Hafler**, on the floor of the exhibit and we had a good time talking about club things. I also met Bob Platt who used to be active in USUS and offered to help us if we needed it. Bob told me his club, Apple Pi, had kicked him out of the club even though he was the president which we both agreed was very strange. He is in a new club now and I hope he does well because he is one of the smartest people I know and also he is a nice guy and not strange at all, at least to me.

I also bought some things on Macintosh disks, and I got some free disks too. When I got back to work I put the disks into our Macintosh and it ate two of them, including my favorite which was a demo disk of "Mathematica" in color which made this sort of three dimensional color graph that looked a little like those plastic boxes that have the ocean or something in them. "Bad, Bad Macintosh" I said, but it was sorry and it was not it's fault as one of it's cables was loose. I forgave it because sometimes my cables are loose too.

We drove back from Boston late at night and William drove off in his truck.

We had a good time and I hope we can go on a trip like this again.

As you can see from above, Harry has a slightly off-beat sense of humor. On a more serious note, one of the topics at the Apple user group breakfast (BTW, Harry wasn't watching and I went back twice. The food was good.) was how to get users to join and participate in the group. We are in a slightly different position from all the rest of the groups at the meeting. One, because we are not strictly a Macintosh group and two because we are not a local group but nation wide (international). Some of the thoughts were the need, not for more techies, but for more business types to run the organization. Volunteer work for fund raisers, charity events, etc. is a good way to get free publicity. To get new members, you have to let people know you are out there.

I found several things of interest. For the Mac, a database program called "4th Dimension" which

I've been using the last couple weeks. A parallel printer card for a Mac II so that I can use my old printer for dumping reports. A backup and archiving program called "Retrospect". This wasn't my first MacWorld so I wasn't as overwhelmed as Harry ☺.

Administrator Says
by Hays Busch

Seems like I'm involved with the Software Library a lot these days. This time, I have a very sophisticated program to tell you about. It is donated to USUS by **David T. Craig** (Mbr. # 4030), is called TIP, and is designed for the IBM PC. As David says, "I realize the USUS IBM PC SIG is not very active. Hopefully this program will change this situation."

What is TIP? The initials stand for The Image Processor and the program allows the user to work with both Terra-Mar and TARGA image files to do image enhancement using various image processing techniques. It runs on any IBM PC, or workalike, which should have an EGA or VGA monitor and must have a minimum of 512K of memory. (NOTE: I tried it on my AT workalike with a Hercules card and monochrome monitor. I could not get the results in color, but could see enough of the program to be very impressed.) The program runs best if the machine has a hard disk and a math co-processor. Without the math chip some operations take longer, but even then the program runs well at 12mhz clock speed. Altho I did not try it, I think it can be run from a 1.2mb floppy.

David wrote the program in Turbo Pascal 4.0. It runs on my machine under MS-DOS 3.2. TIP was his Senior Project at Texas A&M, and David has donated it to USUS for non-commercial use by USUS members and wants it included in the USUS Software Library. We'll be happy to oblige! But we are going to do it on a somewhat exclusive and slightly different basis to start with. Here is how this program will be made available to USUS members for at least the remainder of this calendar year.

The Run Only TIP Package consists of three 5 1/4 inch 360K disks and 18 hardcopy pages of documentation. Disk One contains executable code files for the program. Disk Two and Disk Three disks contain a sample Terra-Mar Image file called ALIEN. (Disk Two and Three need to be "concated" together to create a single image file for the program. That's why a hard disk is a good idea. This is easy to do.) With this package, the program is fully functional. The cost for this package is $10.00 which includes everything mentioned above plus handling and shipping. And this cost is in line with normal SW Libe volume costs.

The Complete TIP Package consists of the above, but adds one more 5 1/4 inch, 360K disk which contains the source code files for TIP. And this is the package I recommend you buy. The cost of this package is $12.00 and if you are at all interested in image processing, you definitely want the source code! As David says, "the program...contains several interesting image processing algorithms...(and)...UCSD users who wish to convert this program to work on other systems should encounter minimal difficulties".

So, how do you order this? Well write me a note and enclose a check made out to USUS, Incorporated for the correct amount of the package you want. Be sure to include your name and mailing address when you write. Mail it all to the LaJolla Post Office address and I'll take it from there. You will be asked to sign the normal USUS Software agreement and mail it back to me as part of the deal. And that requirement is on an "honor code" basis.

Why is this not going into the SW Libe as a normal "Volume series" at this time? Two reasons: One: It is for color (EGA VGA) IBM's and "workalikes" only. Two: We don't have an active IBM SIG to handle the details of getting it into the SW Libe. Thus yours truely is handling it and this is the fastest way to get it to you.

I don't know much about image processing, but I was fascinated by this program. It is well worth its cost even if you (like me) have never seen anything like it before. (Even worth it on my mono monitor, and it must be sensational on a color monitor!) We did not offer a package containing the source code and the image disks because compiling the source is hardly worth the extra $2.00 that you pay for the complete package.

David has provided hardcopy for all the source files of TIP. I can get this to you, but plan to produce it only on an "if ordered" basis. This means you would have to pay the cost of Xeroxing it and the postage involved in mailing it. I have no idea what this will run, but if you want the listings, tell me so in your note and I'll get a price back to you. (There are at least 79 one-sided sheets of 8 1/2 by 11 inch stock involved here so it is a good size copying job. I suspect the cost will be between $8.00 and $15.00 for this.) Also since you can obviously print out the source files on your machine and get a similar listing, I didn't think there'd be much call for this.

In a note from David Craig, he disagrees with Hayes and recommends getting copies of the listings, since they are printed on an Apple LaserWriter and include line cover pages, line numbers, complete cross-references, and a complete list of procedure/function names

NOTE: International Members should add $2.00 to each of the above package prices to cover Airmail handling. And all funds must be in US Dollars on US Banks.

That's about all for this issue. I was impressed by the variety of info in the July/August NewsLetter. All you contributors, keep up the good work! Also, let's hear from some of you who have not contributed so far. Robert Geeslin, our Publisher, apologizes for the late in mail date of that issue. Seems the printer got messed up and delayed the whole process.

The Prez Sez
by Alex Kleider

USUS as an organization is at a critical junction. Its membership, which has I believe at one time been over 5,000, has now dwindled to approximately 250 and all indications are that this trend will continue. Reasons for this are varied. Undoubtedly the fact that the UCSD p-System is no longer THE microcomputer OS is the most important factor. In an effort to circumvent this problem the USUS leadership has been attempting to redefine the organization's purpose to include all aspects of "portable programming". In concrete terms this has meant a greater emphasis on Modula-2 since there is a lot of interest in this language amongst our remaining members and as things stand at the moment (and this may change with the resurrection of MODUS) there is no other organization that caters to this interest. This is not to say that the UCSD p-System is being abandoned. As long as it exists, USUS will continue to include its users.

This brings us to the heart of the issue. Will USUS continue to exist? There has been talk amongst the leadership of closing down and regrettable though it may be, to do so may prove to be the only responsible course of action. The main reason is simple. Not enough member participation. There has been a nucleus of people trying lately to hold the organization together but the same members can not be expected to continue indefinitely to give so heavily of their time. You have all previously read in these pages many pleas for members to get involved in all sorts of ways such as 1. simply providing feed back on what is published in the NewsLetter, 2. submitting code for publication or submission to the Library, 3. taking on the duties of an officer or official. We soon must have an election to replace two members of the Board of Directors so to this list of functions we can add candidacy for the BoD.

In summary then USUS needs two things to survive. One is to maintain its current membership (so renew if your time to do so is up) and build it back up. The other is member participation. Without a volunteer (or perhaps several since the task is great) to take over Hays Busch's work we can not hope to survive after he steps down from his job of USUS administrator. Similarly for the job of editing the NewsLetter, William Smith will not continue this job forever. These represent the biggest jobs and hence the main "crises" but getting effective leadership on the Board of Directors and into other USUS posts is also a necessity. The members of USUS must now stand and be counted not just for their numbers but for what they are willing to contribute to the organization if it is to continue.

Treasurer's Report (July 1989)
by Robert E. Clark, Treasurer

Bank Balance	$5295.56	06-30-89
Income - July 1989		
Dues:		(new/renew)
Student	0.00	0/0
General	420.00	2/10
Professional	100.00	0/1
Institutional	0.00	0/0
Other Income:		
Library fees	12.00	
PowerTools	164.90	
CIS	30.01	
Total Income:	$726.91	
Expenses - July 1989		
Administrator:		
CIS	0.00	
Telephone	18.81	
Photocopies	2.10	
Postage	38.49	
Printing	53.79	
Other:		
Mail from La Jolla	4.25	
Refund	20.00	
Misc.	0.25	
Bank charge	1.00	
Total Expenses	$138.69	
Bank Balance	$5,883.78	07-31-89

Treasurer's Report (August 1989)
by Robert E. Clark, Treasurer

Bank Balance	$5,883.78	07-31-89
Income - August 1989		
Dues:		(new/renew)
Student	0.00	0/0
General	167.50	0/4
Professional	100.00	0/1
Institutional	0.00	0/0
Other Income:		
Library fees	2.00	
CIS	30.02	
Total Income:	$299.52	
Expenses - August 1989		
Administrator:		
CIS	140.81	
Telephone	14.20	
Photocopies	6.60	
Postage	67.89	
Other:		
Mail from La Jolla	8.80	
PowerTools Royalities	204.00	
Bank charge	1.00	
Total Expenses	$443.30	
Bank Balance	$5740.00	08-31-89

Changes in USUS

by Eli Willner, on behalf of the USUS Board of Directors

USUS is in the midst of undergoing some changes. While this organization began as a UCSD Pascal users' group, many of our members —and many of the vendors that sell UCSD Pascal and related products —have broadened their scope. The USUS board of directors feels that is time for USUS to follow suit.

The trick is to broaden our scope without making us a different organization, and while continuing to serve the needs of our current, loyal members. The uniting force that has driven our members through the years has been the focus on software portability. After all, portability has always been the hallmark of the p-System. Our members have always been aware of the need to write software that is independent of hardware and operating system platform.

A few years ago, common wisdom held that portability was no longer a desirable trait in software. IBM had entered the PC market and many felt that it would become the entire computer universe. This has turned out not to be the case, and there is now a renewed and widespread interest in portability.

Portability is now the primary focus of USUS. We want to emphasize that UCSD Pascal will continue to be an important aspect of our focus on portability. But it will not be the only aspect. We have seen a growing interest in Modula-2 in recent months — both the p-System version of Modula-2 and other versions, as well.

Many are relying on C to provide them with the ability to freely move their applications from one environment to another. Whether or not C is the

best medium to achieve portability —and how one does achieve portability using C — will be another area of interest.

MUSUS has a new name: "The Portable Programming Forum". A reorganization of message and software library sections to reflect USUS' new focus is in progress under the stewardship of primary SysOp Harry Baya. There has already been a tremendous influx of technical conversation on issues of portability. Much of this new conversation has been centered around Modula-2. New members are joining MUSUS. When our reorganization is complete, CompuServe will be publicizing the "new MUSUS" and we expect our activity to grow even more. If you have not yet tried MUSUS, or haven't been online for a while, we suggest you try MUSUS again. The CompuServe software underlying our Forum has been made more powerful and easier to use and there are now programs that can dramatically drop the cost of CompuServe usage. You can find out more about these programs once you are online.

Because so many of us have at least some interest in Modula-2, talks are underway with MODUS, the Modula-2 users' group, exploring how our organizations might pool resources and work together.

Because membership in USUS is not necessary in order to access MUSUS, folks who haven't previously heard of us are stopping by, and often staying. When they realize the high level of technical expertise our members enjoy, many of these passers-by become USUS members. Thus, our ranks are enjoying the contributions of "new blood". We expect this to increase once MUSUS reorganization is complete. We hope that these new members will volunteer to serve USUS in various capacities. Especially, we hope that some of these new members will consent to stand for election to our board of directors.

We have postponed elections to the board pending expressions of interest from members, old and new to run. If you would like to run for election to the board of directors, please notify Alex Kleider at (415) 327-7916.

We anticipate and welcome your active participation in the new USUS!

WDS Terminal I/O Unit

By William D. Smith

This is the unit which I use to do all my screen I/O. It is written in a portable manner for a specific terminal (in this case, a Wyse 60 or 50 terminal). To modify it for another terminal, you need to change the `Translate` array, which translates the control characters of the keyboard into the defined constants. You also need to change the `Sc_Cmd` array which writes the different commands to the terminal (such as the clear screen command). You also have to decide how to handle the functions which are not provided by your terminal (for example the info line which need not be used, see next paragraph).

The screen consists of four areas, the terminal status area, the information area, the data area and the message area. The terminal status area is maintained by and used by the terminal. The only part the user can change is the time. This is done by setting the terminal time with the `S_Time` procedure (it writes the time to the terminal which maintains it after that). The information area is at the top right side and is not directly addressable. The `S_Info` procedure writes the string passed to this area along with some other information (the date, `memavail` and `varavail`). Not all terminals have this area. The message area is defined as the last line on the screen. Use `S_Msg` and `C_Msg` to write and erase messages from this area. `Error` and `Err_Msg` also use this line to display errors. The unit maintains a stack of messages which are automatically pushed and popped. The data area is the rest of the screen and is dimensioned `Width x Height`. It is addressed as (0..`Width`-1, 0..`Height`-1).

Output is either control or data. Control procedures such as `Beep` (sound the bell) or `C_Sc` (clear the screen) do something to the terminal. Data procedures (`W_Char`, `W_Str`, `S_Info`, `S_Msg` and `Error`) write information to the terminal.

Input can be in the form of a command preceded by an optional repeatcount (`G_Char`, `G_Chars` or `G_Cmd`) or a string followed by an accept or abort character (`G_Str`). There is also provision for checking to see if a character was typed (`InputWaiting`) or if the user hit the abort key (`UserAbort`)while an operation such as printing was in progress.

The G_Str function has two parameters. On input the string contains the default string displayed and on output it contains the string as entered/modified by the user. Len is the number of characters the user is allowed to entered. When the function starts, it displays the default string and enough underline characters to fill out to Len, leaves the cursor in the first position and enters the start state. In the start state, if the user types a character, the default string is cleared and the character is left at the beginning of the string. If the user types a control character, the string is left as is for modification. The left and right arrows move left and right one character, the tab moves to the end of the string, the back tab moves to the beginning of the string. The get character key followed by a character moves to the next occurrence of that character. The delete character key deletes the current character and moves the rest of the string to the left. The insert character key inserts a space and leaves the cursor in the same place. The insert mode key toggles between insert and exchange mode. The delete key deletes from the cursor to the end of the field. The clear line key clears the whole field. The retry key restarts the function with the default string. The escape key terminates the function, returning the default string. The accept keys terminate the function returning the entered string. Both the escape and accept keys are returned as the function value. This is useful when you have a page of information and want to use the accept key to signify where to go next. For example accept and go to the beginning of the next line instead of the next field.

This unit supports 16 user definable function keys. "^R" followed by the function key records everything typed until the next "^R". You may nest the function keys. The only limit is the size of the function key buffer (456 characters). You may also save up to 10 sets of function keys. Of the 16 function keys, only 15 are saved. The date key is reset to the system date every time the unit is initialized.

The figure shows a sample screen for a typical application. Since the Wyse terminals use embedded character attributes, the prompts and message line have a space at both ends. This is the attribute character. Py is incremented, decremented or set (as in this case) to where you want the prompt line to appear. Always restore it to its previous value.

Note: The cursor status, keyboard lock status and messgae lines are maintained on a stack.

```
0 Terminal Status line      00:00 Addressable Info area
1  Prompt line: C(opy D(isplay Q(uit                          [1.00]
2
3    S = Srource file ---> Sample.Text
4    T = Target file ---->
5
6
7
8
9
10
11  Display: P(age O(ppsitePage Q(uit                          [1.00]
12 {$P}
13   procedure ReadFile (InFile : Str_23);
14   var  StdIn : text;
15   begin
16     readln (StdIn, InFile)
17
18     while not (eof (StdIn)) do  begin
19       { process the file }
20     end { while };
21   end { ReadFile };
22
23
24 begin { main }
25  Message line, <space> to continue ■
```

```
{ WDS terminal input / output unit [1.16] --- 02 Aug 88 } { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1987 to 1988,  All rights reserved.                }
{ File:         T_Io_U.Text                Version 1.16    02 Aug 88

  Author:       William D. Smith           Phone: (619) 941-4452
                P.O. Box 1139              CIS: 73007,173
                Vista, CA  92083

  Notice:       The information in this document is the exclusive
                property of William D. Smith.  All rights reserved.
                Copyright (c) 1987 to 1989.

  System:       Power System version IV.2.2

  Compiler:     Power System Pascal Compiler

  Keywords:     WDS T_Io_U Terminal Input Output Unit

  Description:  WDS terminal input / output unit.  This unit contains procedures
                to handle all terminal I/O.

  Change log: (most recent first)

Date        Id   Vers  Comment
---------   ---  ----  -------
02 Aug 88   WDS  1.16  In Initialize, called Set_ScSize to init V_, H_Size.
20 Mar 88   WDS  1.15  Put varavail in info line.
09 Mar 88   WDS  1.14  Added checks for sting overflow and Reverse to S_Prompt.
06 Feb 88   WDS  1.13  Added NoMsg to C_Sc and C_Eop.  Moved M_ to Glbs_U.
28 Jan 88   WDS  1.12  Fixed so that Dbk can be entered in UserAbort.
30 Dec 87   WDS  1.11  Fixed error in S_Time when time is blank.
20 Oct 87   WDS  1.10  Fixed S_Time, changed values of clear commands.
23 Sep 87   WDS  1.09  Added V_Size and H_Size.
16 Sep 87   WDS  1.08  Added several procedures.
31 Aug 87   WDS  1.07  Fixed for version IV.22.
20 Aug 87   WDS  1.06  Added UserAbort.
16 Jul 87   WDS  1.05  Put in version control.
19 Jun 87   WDS  1.04  Used Cap from StrOps_U.  Updated X, Y in C_Sc.
06 Jun 87   WDS  1.03  Added Prompt to S_Msg, Map to G_Char, fixed some errors.
05 Jun 87   WDS  1.02  Added G_Char_Answer.  Removed Ins/DelLine, Bs and Fs.
29 May 87   WDS  1.01  Finished conversion.
08 May 87   WDS  1.00  From FDB2 T_Io_U [2.21].
}
{$I VERSION.TEXT}  { Declares conditional compilation flags }
{$D CSR-}  { Outputs cursor type }
{$D KBD-}  { Outputs keyboard lock status }
unit T_Io_U;
interface {$ T_Io_U [1.16] 02 Aug 88 Wyse 60 }
uses  Glbs_U;   { WDS globals unit }
const Vc_T_Io_U    =  8;   { 09 Mar 88 }
      Vs_T_Io_U    = 'T_Io_U';

      Nul          =  0;   { ^@, null character }

      Esc          =  1;   { abort, don't move }

      Acc          =  2;   { accept, don't move }
      Anf          =  3;   { accept, goto next field }
      Apf          =  4;   { accept, goto previous field }
      Anl          =  5;   { accept, goto next line }
      Apl          =  6;   { accept, goto previous line }
      Ach          =  7;   { accept, goto start<Home> }
```

```
        Uk              =  8;   { up key }
        Dk              =  9;   { down key }
        Lk              = 10;   { left key }
        Rk              = 11;   { right key }

    { Keys 12 to 31, and 127 are reserved for internal use. }

type    Sc_Size         = (SameSc,    { Set Width or Height, don't adjust screen }
                           AdjustSc,  { Adjust screen to match Width or Height }
                           SmallSc,   { Small width or height }
                           XSmallSc,  { Alternate small width or height, if any }
                           LargeSc,   { Large width or height }
                           XLargeSc   { Alternate large width or height, if any }
                          );

        Sc_Color        = (LightSc,   { Black on white }
                           DarkSc     { White on black }
                          );

        CrtTypes        = (Unknown, Wyse50, Wyse60);

var     Vv_T_Io_U       : integer;
        X               : integer;  { r/o, current column on the screen }
        Y               : integer;  { r/o, current line on the screen }
        Py              : integer;  { r/w, current prompt line, inc/dec as needed }
        Width           : integer;  { r/o, width of the screen, 1 based }
        Height          : integer;  { r/o, height of the screen, 1 based }
        V_Size          : Sc_Size;  { r/o, vertical size, SmallSc..XLargeSc }
        H_Size          : Sc_Size;  { r/o, horizontal size, SmallSc..XLargeSc }
        Color           : Sc_Color; { r/o, color of screen }
        CrtType         : CrtTypes; { r/o, supported CRT types }

{ The screen consists of three areas, the information area, the data area and
  the message area.  The information area is at the top right side and is not
  directly addressable (use S_Info).  The message area is the last line on the
  screen (use S_Msg and C_Msg).  The data area is the rest of the screen and is
  dimensioned Width x Height.  It is addressed as (0..Width-1, 0..Height-1).
  All characters returned by the input procedures are in the range 0..127 (7
  bits).
 }

{ Screen control procedures }

  procedure Beep;
  { Ring the terminal bell. }

  procedure SetXy (Xx, Yy : integer);
  { Set the cursor position to Xx and Yy.  If Xx and/or Yy are Null, they are
    set to zero.  Put the cursor at the position of Xx and Yy and update the
    global values of X and Y.  No checking is done.
   }

  procedure Set_Sc_Size (Vert, Horz : Sc_Size);
  { Set screen size. }

  procedure Set_Sc_Color (Color : Sc_Color);
  { Set_Sc_Color.  This procedures set the screen color. }

  procedure C_Sc (NoMsg : boolean);
  { Clear the screen.  The cursor is left at the home position (0, 0).  If NoMsg
    is true, the message line is also cleared.  This is used for writing to the
    console as if were a file (ie. using Tx_Io_U).
   }
```

```
  procedure C_Eop (NoMsg : boolean);
  { Clear to end of page from the current cursor position.  The cursor is not
    moved.  If NoMsg is true, the message line is also cleared.  This is used
    for writing to the console as if were a file (ie. using Tx_Io_U).
  }

  procedure C_Eol;
  { Clear to end of line from the current cursor position.  The cursor is not
    moved.
  }

  procedure C_Eof (N : integer);
  { Clear to end of field.  N defines the distance of the end of field from the
    current cursor position.  The cursor is left at the begining of the field
    (ie. the cursor is not moved).
  }

  procedure S_Rev (Len : integer);
  { Show reverse.  Len spaces on the screen begining at the cursor position are
    reversed.  The cursor is moved forward one character position.
  }

  procedure S_Dim (Len : integer);
  { Show dim.  Len spaces on the screen begining at the cursor position are
    dimmed.  The cursor is moved forward one character position.
  }

  procedure S_Unln (Len : integer);
  { Show underline.  Len spaces on the screen begining at the cursor position
    are underlined.  The cursor is moved forward one character position.
  }

  procedure S_RevDim (Len : integer);
  { Show reverse and dim.  Len spaces on the screen begining at the cursor
    position are reversed and dimed.  The cursor is moved forward one character
    position.
  }

  procedure C_Atr (Len : integer);
  { Clear the attribute.  Len spaces on the screen begining at the cursor
    position are clear of their attribute (if any).  The cursor is moved forward
    one character position.
  }

  procedure Cursor (Want : OnOff);
  { Cursor display control.  The cursor is turned on (Want = On), turned off
    (Want = Off), toggled (Want = Toggle), or reset to the previous value (Want
    = Pop).  (Want = Show) readjusts the cursor.
  }

  procedure LockKeyboard (Want : OnOff);
  { Lock keyboard.  The keyboard is locked (Want = On), unlocked (Want = Off),
    toggled (Want = Toggle), or reset to the previous value (Want = Pop).  (Want
    = Show) readjusts the keyboard.
  }

{ Screen output procedures.  For each of the following procedures, the output is
  written to the screen at the current cursor position, the cursor is left at
  the end of the output and the global value of X is updated to reflect the new
  cursor position.  The output does not wrap to the next line.
}

  procedure W_Str (S : Str_255);
  { Write the string S to the screen. }

  procedure W_Char (C : char);
  { Write the character C to the screen. }
```

```
{ Miscellaneous procedure. }

  function Ck_Accept (Ch : char) : boolean;
  { Check accept.  This function returns true if the character is one of the
    accept characters.  It returns false otherwise.
  }

  procedure FlushKeyboard;
  { This procedure throws away all input typed ahead at the keyboard.  It also
    cancels function execution.
  }

{ Keyboard input procedures.  Chars must not contain lowercase letters. }

  function G_Char (Chars : CharSet;  Flush, Map : boolean) : char;
  { Get a character.  This function reads the keyboard until a character in the
    set of characters Chars is typed.  If Flush is true, all characters already
    typed and not read are thrown away (including characters in a currently
    executing function key).  If Map is true, lowercase are mapped to uppercase
    characters.
  }

  function G_Chars (Chars : CharSet;  Map : boolean) : char;
  { Get a character or control key.  This function reads the keyboard until a
    character in the set of characters Chars or a control key is typed.  If Map
    is true, lowercase are mapped to uppercase characters.
  }

  function G_Cmd (Chars : CharSet;  var Count : integer) : char;
  { Get command.  This function reads the keyboard until a character in the set
    of characters Chars is typed.  Count is returned as a valid positive number
    typed immediately preceding a valid command character.  If Infinity ('/') is
    typed, Count is returned as Null.  A Cancel ('.') cancels the repeat count.
    Cancel and Infinity can not be in Chars.  This procedure is case
    insensitive.
  }

  function G_Str (var S : Str_255;  Len : integer) : char;
  { Get string.  The user is allowed to enter a string up to Len characters
    long.  S initially contains the default string.  The function returns the
    terminating control character.
  }

  function InputWaiting : boolean;
  { This function returns true if there is a character waiting to be read from
    the keyboard.  The character is not read.
  }

  function UserAbort : boolean;
  { This function returns true if the abort key was typed.  It returns false if
    nothing was typed or if any key besides the abort key was typed.
  }

{ Prompt, information, message, and error procedures }

  function S_Prompt (Prompt  : Str_255;  Version : Str_7;
                     Reverse : boolean) : integer;
  { Show prompt.  The prompts are written on line Py (global).  Prompt is the
    string written and Version is a string written on the far right side of the
    screen.  If Reverse is true, the prompt is shown with a S_Rev attribute.
    The first and last column on the line are used for the attributes.  The
    function returns the X location of the end of the prompt string (where the
    cursor would normally be plus one space).  If Prompt won't fit on the
    screen, it is truncated.
  }
```

```
  procedure S_Info (S : Str_255);
  { Write information line.  This procedure displays the string S on the
    information line on the screen.  If the string is too long, it is truncated.
    It also displays the date, memavail and varavail.
  }

  procedure S_Time;
  { This procedure sets the terminals time. }

  procedure S_Msg (Prompt, Attention : boolean;  S : Str_255);
  { Show message.  If Attention, the message line is blinking and the terminal
    bell is rung once.  If Prompt, the cursor is left at the end of the message.
  }

  procedure C_Msg (Msgs : integer);
  { Clear messages.  Msgs tells how many messages to clear (0 clears all). }

  function G_Char_Answer (Chars : Char_Set;  Flush : boolean;
                          Attention : boolean;  M : Str_255) : char;
  { Get character after showing message.  This function displays the message M,
    reads the keyboard until a character in the set of characters Chars is
    typed, and then clears the message.
  }

  procedure Error (S : Str_255);
  { Error.  This procedure displays the string S on the message line on the
    screen and waits for the user to type a space.
  }

  procedure Err_Msg (Msg : integer;  S : Str_255);
  { Error message.  This procedure displays a message cooresponding to Msg on
    the screen message line and waits for the user to type a space.  S is
    appended to the message.  If the error is an I/O error, S should be the
    filename the error occured on.
  }

  procedure Set_ErrFiles (SysErr, UserErr, SoftKeys : Str_23);
  { Set error file names.  This procedure sets the name of the error files.  If
    the strings are empty, the names are not changed.  The default names are
    'WDS.ERRS', 'WDS.ERRS', and 'WDS.KEYS'.
  }

implementation

uses  StrOps_U,          { WDS string conversion ops unit }
      F_Io_U,            { WDS file I/O unit }
      OpSys_U;           { WDS to operating system interface unit }

const Bel       = 7;     { bell }
      Eol       = 13;    { return }
      U_Kbd_Ch  = 14;    { unlock keyboard character }
      L_Kbd_Ch  = 15;    { lock keyboard character }
      C_Sc_Ch   = 26;    { clear screen character (Wyse 60) }
      Pki       = 27;    { prefix key from keyboard (Wyse 60) }
      Pko       = 27;    { prefix key to screen (Wyse 60) }

    { Field editing keys 12, 13, 14, 15 not used }

      Tk        = 16;    { Tab key, move to end of field }
      Btk       = 17;    { BackTab key, move to start of field }
      Ick       = 18;    { Insert char key }
      Dck       = 19;    { Delete char key }
      Imk       = 20;    { Insert mode key }
      Dmk       = 21;    { Delete mode key }
      Del       = 22;    { Clear to end of line key }
      Clk       = 23;    { Clear line key }
      Rtk       = 24;    { ReTry key }
```

```
    Gck         = 25;    { Get char key }
    Gak         = 26;    { Get again key }
    Cfk         = 27;    { Cancel fist key }

  { Miscellanous keys }

    Ftk         = 28;    { Flush toggle key }
    Rcd         = 29;    { Record toggle key }
    Dbk         = 30;    { Debug key }
    Ofk         = 31;    { Read/write function keys }

    Infinity    = '/';   { Repeat count = Null in G_Cmd }
    Cancel      = '.';   { Cancels repeat count in G_Cmd }

    Fcn_D_Len = 8;       { Length of the Date in chars }
    Fcn_Date  = 446;     { Start location of the current date }
    Fcn_Max   = 453;     { Last entry in function buffer }
    Fcn_F_Key = 128;     { First function key }
    Fcn_L_Key = 143;     { Last function key }
    P_Bias    = 128;     { Prefixed key bias }

    Console   = 1;
    SysTerm   = 2;

    H_Small   = 80;      { Horizontial screen size, X = extra }
    H_XSmall  = 80;
    H_Large   = 132;
    H_XLarge  = 132;

    V_Small   = 24;      { Vertical screen size, X = extra }
    V_XSmall  = 25;
    V_Large   = 42;
    V_XLarge  = 43;

    Msg_File  = 'WDS.ERRS';      { Name of the error message file }
    Key_File  = 'WDS.KEYS';      { Name of the soft function key file }

    Max_Csr   = 15;      { Maximum depth of the cursor stack }
    Max_Kbd   = 15;      { Maximum depth of the keyboard stack }
    Min_Msg   = 0;
    Max_Msg   = 3;       { Maximum number of messages in the stack }
    Msg_StrSz = 81;      { Maximum length of Msg string, Make Msg_S agree }

    TaMin     = 0;       { Type ahead stuff }
    TaMax     = 15;

type  Fcn_Rec    = record  { 2 words }
                     Fcn_First   : integer;  { Beginning of function defn }
                     Fcn_Last    : integer;  { End of function defn }
                   end { Fcn_Rec };

    Sc_Cmmd    = packed array [0..3] of char;

    Sc_Cmds    = (Sc_Beep,          { Sound the bell }
                  Sc_C_Sc,          { Clear the screen }
                  Sc_C_Eop,         { Clear to the end of page }
                  Sc_C_Eol,         { Clear to the end of line }
                  Sc_C_Atr,         { Clear attribute }
                  Sc_Rev_Atr,       { Reverse attribute }
                  Sc_Dim_Atr,       { Dim attribute }
                  Sc_Unln_Atr,      { Undeline attribute }
                  Sc_RevDim,        { Reverse and Dim attribute }
                  Sc_Norm_Atr,      { Normal attribute }
                  Sc_S_Csr,         { Show the cursor }
                  Sc_C_Csr,         { Clear (hide) the cursor }
                  Sc_U_Kbd,         { Unlock the keyboard }
                  Sc_L_Kbd,         { Lock the keyboard }
```

```
                    Sc_S_Horz,      { Small horzontial screen (80 columns) }
                    Sc_L_Horz,      { Large horzontial screen (132 columns) }
                  { Sc_Xs_Horz, }   { Extra & small horzontial (not used) }
                  { Sc_Xl_Horz, }   { Extra & large horzontial (not used) }
                    Sc_S_Vert,      { Small vertical (24 rows) }
                    Sc_L_Vert,      { Large vertical (42 rows) }
                    Sc_Xs_Vert,     { Extra & small vertical (25 rows) }
                    Sc_Xl_Vert,     { Extra & large vertical (43 rows) }
                    Sc_Light,       { Light screen, dark characters }
                    Sc_Dark         { Dark screen, light characters }
                   );

    Msg_Rec    = packed record  { 42 words }
       8 bits }    Old_X  : Byte;      { X loc before S_Msg }
       6 bits }    Old_Y  : 0..63;     { Y loc before S_Msg }
       1 bit  }    Msg_P  : boolean;   { Message prompt }
       1 bit  }    Msg_A  : boolean;   { Message attention }
      41 words}    Msg_S  : Str_81;    { Make Msg_StrSz agree }
                  end { Msg_Rec };

var   Flush_On   : boolean;  { Controls type ahead flushing and async reads }
      OnlyOne    : boolean;  { Used by UserAbort and Next_Char }
      Csr_Stack  : packed array [0..Max_Csr] of boolean;  { Cursor stack }
      Csr_Tos    : integer;  { Cursor top of stack }

      Kbd_Stack  : packed array [0..Max_Kbd] of boolean;  { keyboard stack }
      Kbd_Tos    : integer;  { Keyboard top of stack }

      ReverseSc  : boolean;  { Screen is reversed }
      OneErrFile: boolean;  { Sys and User have the same filename }

      UseTa      : boolean;  { Use type ahead buffer }
      TaIdx      : integer;  { Index into type ahead buffer }
      TaLast     : integer;  { Last valid character in type ahead buffer }
      Ta         : packed array [TaMin..TaMax] of char;

      Fcn_Ex     : boolean;  { Executing a function? }
      Fcn_Rcd    : boolean;  { Recording a function? }
      Fcn_Ch     : boolean;  { Char came from a function, not input }
      Fcn_K_Ex   : integer;  { Function being executed }
      Fcn_K_Rcd  : integer;  { Function being recorded }
      Fcn_I_Ex   : integer;  { Index in Fcn_Buf of function being executed }
      Fcn_Stop   : integer;  { To catch infinite loops }
      In_Ofk     : boolean;  { Reading/writing function keys }

    { The following data is read from a file.  Do not change w/o changing file.
      The first function key (F0, 2 words) and the last 8 bytes (4 words) are not
      read or written to the function key file.  They always contain the current
      date.  Only 256 words are read or written, from Fcn_Keys [1] to Fcn_Buf
      [445].
    }
      Fcn_Keys   : array [Fcn_F_Key..Fcn_L_Key] of Fcn_Rec;    {  32 words }
      Fcn_I_Rcd  : integer;  { Next loc to record into }       {   1 word  }
      Fcn_Buf    : packed array [0..Fcn_Max] of char;          { 228 words }
    { up to here }                                             { 261 words }

      Translate  : packed array [0..255] of 0..255;
      Sc_Cmd     : packed array [Sc_Cmds] of Sc_Cmmd;

      Smsg_File  : Str_23;     { WDS system error message filename }
      Umsg_File  : Str_23;     { User error message filename }
      Fcn_File   : Str_23;     { Soft function key filename }

      Init_Msg   : Str_5;      { String written to terminal to start a msg }
      InfoWidth  : integer;    { Width of info line (user part) }
```

```
    Info       : Str_31;     { Displayed by S_Info }
    InfoStr    : Str_81;     { What is displayed currently }

    In_G_Str   : boolean;    { In G_Str on Msg line }
    Gtmp       : Str_255;    { Work string for G_Str }

    L_Chars    : CharSet;    { Legal display chars }
    C_Chars    : CharSet;    { Control characters }
    S_Chars    : CharSet;    { String input/editing characters }

    Msg_X      : integer;    { Message column number }
    Msg_Y      : integer;    { Message line number }
    Clr_Msg    : boolean;    { Used only by S_Msg, C_Msg and Ck_Msg }
    Msg_Tos    : integer;    { Msg top of stack (next location to use) }
    Msg_Stack  : array [Min_Msg..Max_Msg] of Msg_Rec;

procedure Control (Sc : Sc_Cmds);  forward;

procedure Set_Info;  forward;

procedure Ck_Msg;  forward;

segment procedure Initialize;
var Ss : Str_23;
  procedure Init_Translate;  { Wyse 60 }
  var I : integer;
  begin
    fillchar (Translate, sizeof (Translate), Nul);
    for I := ord (' ') to ord ('~') do  Translate [I] := I;

    Translate [  3] := Acc;    { ^C }
    Translate [  4] := Dbk;    { ^D }

    Translate [  7] := Gck;    { ^G }
    Translate [  8] := Lk;     { ^H or <backspace> }
    Translate [  9] := Tk;     { ^I or <tab> }
    Translate [ 10] := Dk;     { ^J or <down> }
    Translate [ 11] := Uk;     { ^K or <up> }
    Translate [ 12] := Rk;     { ^L or <right> }
    Translate [ 13] := Anl;    { ^M or <return> }

    Translate [ 18] := Rcd;    { ^R }

    Translate [ 26] := Dbk;    { ^Z }
    Translate [ 29] := Lk;     { ^] or <left> }

    Translate [ 30] := Ach;    { ^shift~ or <home> }
    Translate [ 31] := Ofk;    { ^_ or ^<Del> }

    Translate [127] := Del;

    Translate [128] := Fcn_F_Key + 8;  { Ctrl F1 }
    Translate [129] := Fcn_F_Key + 9;
    Translate [130] := Fcn_F_Key + 10;
    Translate [131] := Fcn_F_Key + 11;
    Translate [132] := Fcn_F_Key + 12;
    Translate [133] := Fcn_F_Key + 13;
    Translate [134] := Fcn_F_Key + 14;
    Translate [135] := Fcn_F_Key + 15; { Ctrl F8 }

    Translate [155] := Esc;    { <esc><esc> or <F12> }

    Translate [161] := Fcn_F_Key + 0;  { F1 }
    Translate [162] := Fcn_F_Key + 1;
    Translate [163] := Fcn_F_Key + 2;
    Translate [164] := Fcn_F_Key + 3;
    Translate [165] := Fcn_F_Key + 4;
    Translate [166] := Fcn_F_Key + 5;
    Translate [167] := Fcn_F_Key + 6;
    Translate [168] := Fcn_F_Key + 7;  { F8 }
```

```
    Translate [183] := Anf;    { <esc>7 or <send> }
    Translate [197] := Imk;    { <esc>E or <INS Line> }
    Translate [199] := Gak;    { <esc>G }
    Translate [201] := Btk;    { <esc>I or shift<Tab> }
    Translate [202] := Apl;    { <esc>J or <PAGE Prev> }
    Translate [203] := Anl;    { <esc>K or <PAGE Next> }
    Translate [208] := Apf;    { <esc>P or <Print> }
    Translate [209] := Ick;    { <esc>Q or <INS Char> }
    Translate [210] := Dmk;    { <esc>R or <DEL Line> }
    Translate [212] := Clk;    { <esc>T or <CLR Line> }
    Translate [215] := Dck;    { <esc>W or <DEL Char> }
    Translate [231] := Gak;    { <esc>g }
    Translate [241] := Cfk;    { <esc>q or <Ins> }
    Translate [242] := Rtk;    { <esc>r or <Repl> }
    Translate [254] := Ftk;    { ^shift<Del>  }
  end { Init_Translate };

  procedure Init_Sc_Cmd;  { Wyse 60 }
  begin
    fillchar (Sc_Cmd, sizeof (Sc_Cmd), 0);
{1}  Sc_Cmd [Sc_Beep,     0] := chr (Bel);
{2}  Sc_Cmd [Sc_C_Sc,     0] := chr (Pko);  Sc_Cmd [Sc_C_Sc,     1] := '*';
{2}  Sc_Cmd [Sc_C_Eop,    0] := chr (Pko);  Sc_Cmd [Sc_C_Eop,    1] := 'y';
{2}  Sc_Cmd [Sc_C_Eol,    0] := chr (Pko);  Sc_Cmd [Sc_C_Eol,    1] := 't';
{1}  Sc_Cmd [Sc_C_Atr,    0] := ' ';
{3}  Sc_Cmd [Sc_Rev_Atr,  0] := chr (Pko);  Sc_Cmd [Sc_Rev_Atr,  1] := 'G';
                                            Sc_Cmd [Sc_Rev_Atr,  2] := '0';
{3}  Sc_Cmd [Sc_Dim_Atr,  0] := chr (Pko);  Sc_Cmd [Sc_Dim_Atr,  1] := 'G';
                                            Sc_Cmd [Sc_Dim_Atr,  2] := 't';
{3}  Sc_Cmd [Sc_Unln_Atr, 0] := chr (Pko);  Sc_Cmd [Sc_Unln_Atr, 1] := 'G';
                                            Sc_Cmd [Sc_Unln_Atr, 2] := '<';
{3}  Sc_Cmd [Sc_RevDim,   0] := chr (Pko);  Sc_Cmd [Sc_RevDim,   1] := 'G';
                                            Sc_Cmd [Sc_RevDim,   2] := 'p';
{3}  Sc_Cmd [Sc_Norm_Atr, 0] := chr (Pko);  Sc_Cmd [Sc_Norm_Atr, 1] := 'G';
                                            Sc_Cmd [Sc_Norm_Atr, 2] := '4';
{4}  Sc_Cmd [Sc_S_Csr,    0] := chr (Pko);  Sc_Cmd [Sc_S_Csr,    1] := '`';
                                            Sc_Cmd [Sc_S_Csr,    2] := '1';
{$B CSR+}                                   Sc_Cmd [Sc_S_Csr,    3] := '+';
{$E CSR+}
{4}  Sc_Cmd [Sc_C_Csr,    0] := chr (Pko);  Sc_Cmd [Sc_C_Csr,    1] := '`';
                                            Sc_Cmd [Sc_C_Csr,    2] := '0';
{$B CSR+}                                   Sc_Cmd [Sc_C_Csr,    3] := '-';
{$E CSR+}
{2}  Sc_Cmd [Sc_U_Kbd,    0] := chr (U_Kbd_Ch);
{2}  Sc_Cmd [Sc_L_Kbd,    0] := chr (L_Kbd_Ch);
{$B KBD+}
     Sc_Cmd [Sc_U_Kbd,    1] := '-';
     Sc_Cmd [Sc_L_Kbd,    1] := '+';
{$E KBD+}
{3}  Sc_Cmd [Sc_S_Horz,   0] := chr (Pko);  Sc_Cmd [Sc_S_Horz,   1] := '`';
                                            Sc_Cmd [Sc_S_Horz,   2] := ':';
{3}  Sc_Cmd [Sc_L_Horz,   0] := chr (Pko);  Sc_Cmd [Sc_L_Horz,   1] := '`';
                                            Sc_Cmd [Sc_L_Horz,   2] := ';';
{3}  Sc_Cmd [Sc_S_Vert,   0] := chr (Pko);  Sc_Cmd [Sc_S_Vert,   1] := 'e';
                                            Sc_Cmd [Sc_S_Vert,   2] := '(';
{3}  Sc_Cmd [Sc_L_Vert,   0] := chr (Pko);  Sc_Cmd [Sc_L_Vert,   1] := 'e';
                                            Sc_Cmd [Sc_L_Vert,   2] := '*';
```

```
{3}   Sc_Cmd [Sc_Xs_Vert,  0] := chr (Pko);  Sc_Cmd [Sc_Xs_Vert,  1] := 'e';
                                             Sc_Cmd [Sc_Xs_Vert,  2] := ')';
{3}   Sc_Cmd [Sc_Xl_Vert,  0] := chr (Pko);  Sc_Cmd [Sc_Xl_Vert,  1] := 'e';
                                             Sc_Cmd [Sc_Xl_Vert,  2] := '+';
{4}   Sc_Cmd [Sc_Light,    0] := chr (Pko);  Sc_Cmd [Sc_Light,    1] := 'A';
                                             Sc_Cmd [Sc_Light,    2] := '0';
                                             Sc_Cmd [Sc_Light,    3] := '4';
{4}   Sc_Cmd [Sc_Dark,     0] := chr (Pko);  Sc_Cmd [Sc_Dark,     1] := 'A';
                                             Sc_Cmd [Sc_Dark,     2] := '0';
                                             Sc_Cmd [Sc_Dark,     3] := '0';
  end { Init_Sc_Cmd };

  procedure Init_Fcns;
  var Tad : TadRec;  Ss : Str_9;
  begin
    Fcn_Ch := false;          Fcn_Rcd := false;
    Fcn_Ex := false;          Fcn_I_Rcd := 0;
    Fcn_I_Ex := 0;            Fcn_K_Rcd := Fcn_F_Key;
    Fcn_K_Ex := Fcn_F_Key;
    In_Ofk := false;          Fcn_Stop := 0;
    fillchar (Fcn_Keys, sizeof (Fcn_Keys), 255 { Null });
    fillchar (Fcn_Buf, sizeof (Fcn_Buf), chr (Nul));
    Get_Sys_Tad (Tad);
    Ss := '00-00-00';
    D_into_S (Tad .D, Ss, 1);
    moveleft (Ss [1], Fcn_Buf [Fcn_Date], length (Ss));
    with Fcn_Keys [Fcn_F_Key + 7] do begin
      Fcn_First := Fcn_Date;
      Fcn_Last := Fcn_Max;
    end { with };
  end { Init_Fcns };

  procedure Init_Msgs;  { Wyse 60 }
  { Height and Width must be initialized before calling this procedure. }
  var S : Str_5;
  begin
    Msg_X := 0;               Msg_Y := Height;
    Clr_Msg := false;         Msg_Tos := Min_Msg - 1;
                {1234}
    Init_Msg := '*G4 ';       Init_Msg [1] := chr (Pko);
            {12345678 1 2345678 2 2345678 3 }
            {      *10 00-00-00 00000 00000 }
    Info := '*A30*F                         ';
    Info [1] := chr (Pko);    Info [5] := chr (Pko);
    InfoWidth := Width - 59;  { Wyse 60 }
    InfoStr := '';
    Set_Info;
  end { Init_Msgs };

  procedure Init_CrtType;
  var I     : integer;
      Ch    : packed array [0..1] of char;
      S     : Str_81;
      Words : array [0..29] of integer;
  begin
    LockKeyboard (On);
    unitstatus (SysTerm, Words, 1);
```

```
    if Words [0] > 0 then
      begin
        UseTa := true;
        TaIdx := TaMin;          TaLast := Words [0] - 1;
        if TaLast > TaMax then  TaLast := TaMax;
        unitread (SysTerm, Ta [TaMin], TaLast + 1);
        unitclear (SysTerm);
      end { if };
    W_Char (chr (Pko));  { ask for terminal type }
    W_Char (' ');
    I := 0;
    unitread (SysTerm, Ch [0], 1);
    while Ch [0] <> chr (Eol) do begin
      I := I + 1;
      S [I] := Ch [0];
      unitread (SysTerm, Ch [0], 1);
    end { while };
    S [0] := chr (I);
    LockKeyboard (Pop);
    if S = '50' then  CrtType := Wyse50
    else if S = '60' then  CrtType := Wyse60
    else  CrtType := Unknown;
  end { Init_CrtType };
begin { Initialize }
  Vv_T_Io_U := Vc_T_Io_U;
  Ck_Version (Vv_Glbs_U, Vc_Glbs_U, Vs_T_Io_U, Vs_Glbs_U);
  Ck_Version (Vv_StrOps_U, Vc_StrOps_U, Vs_T_Io_U, Vs_StrOps_U);
  Ck_Version (Vv_F_Io_U, Vc_F_Io_U, Vs_T_Io_U, Vs_F_Io_U);
  Ck_Version (Vv_OpSys_U, Vc_OpSys_U, Vs_T_Io_U, Vs_OpSys_U);
  L_Chars := [' '..'~'];
  C_Chars := [chr (Esc), chr (Acc),
              chr (Anf), chr (Apf), chr (Anl), chr (Apl), chr (Ach),
              chr (Lk),  chr (Rk),  chr (Dk),  chr (Uk)];
  S_Chars := L_Chars +
             C_Chars +
             [chr (Tk),  chr (Btk), chr (Gak), chr (Gck),
              chr (Ick), chr (Dck), chr (Imk), chr (Dmk),
              chr (Rtk), chr (Del), chr (Clk), chr (Cfk)];
  Set_ErrFiles (Msg_File, Msg_File, Key_File);
  OnlyOne := false;  { used by Next_Char and UserAbort }
  Flush_On := true;
  Csr_Tos := 0;                Csr_Stack [Csr_Tos] := true;
  Kbd_Tos := 0;                Kbd_Stack [Kbd_Tos] := false;
  Width := Get_Sc_Width;       Height := Get_Sc_Height - 1;
  UseTa := false;
  Init_Sc_Cmd;                 Init_CrtType;
  Init_Translate;              Init_Fcns;
  Init_Msgs;
  Py := 0;                     ReverseSc := true;
  C_Sc (false);                S_Time;
  Set_Sc_Size (SameSc, SameSc);
  Control (Sc_S_Csr); Control (Sc_U_Kbd);
end { Initialize };
```

```
procedure Control { (Sc : Sc_Cmds) };
begin
  unitwrite (Console, Sc_Cmd [Sc], sizeof (Sc_Cmmd), , 12);
end { Control };
procedure Beep;
begin
  Control (Sc_Beep);
end { Beep };
procedure SetXy { (Xx, Yy : integer) };
begin
  if (X <> Xx) or (Y <> Yy) then
    begin
      if Xx = Null then  X := 0
      else  X := Xx;
      if Yy = Null then  Y := 0
      else  Y := Yy;
      gotoxy (X, Y);
    end { if };
end { SetXy };
procedure Delay;
var I : integer;
begin
  for I := 0 to 300 do ;
end { Delay };
procedure Set_Sc_Color { (Color : Sc_Color) };
{ ?? need to adjust atributes }
begin
  if Color = LightSc then  Control (Sc_Light)
  else { if Color = DarkSc then }  Control (Sc_Dark);
  ReverseSc := Color = LightSc;
end { ReverseSc };
procedure Set_Sc_Size { (Vert, Horz : Sc_Size) };
var I, J : integer;  S : Str_9;
begin
  C_Sc (true);
  I := Height;
  case Vert of
    SameSc,
    AdjustSc : begin
                 Height := Get_Sc_Height;
                 if (Height <> V_Small) and (Height <> V_Large) and
                    (Height <> V_XSmall) and (Height <> V_XLarge) then
                   begin
                     Height := V_Small;
                     Vert := AdjustSc;
                   end { if };
               end { if };
    SmallSc  : Height := V_Small;
    XSmallSc : Height := V_XSmall;
    LargeSc  : Height := V_Large;
    XLargeSc : Height := V_XLarge;
  end { cases };
  if Height = V_Small then  V_Size := SmallSc
  else if Height = V_XSmall then  V_Size := XSmallSc
  else if Height = V_Large then  V_Size := LargeSc
  else { if Height = V_XLarge then }  V_Size := XLargeSc;
```

```
if Vert > SameSc then
  begin
    case V_Size of
      SmallSc  : Control (Sc_S_Vert);
      XSmallSc : Control (Sc_Xs_Vert);
      LargeSc  : Control (Sc_L_Vert);
      XLargeSc : Control (Sc_Xl_Vert);
    end { cases };

    Delay;
  end { if };

case Horz of
  SameSc,
  AdjustSc : begin
               Width := Get_Sc_Width;
               if (Width <> H_Small) and (Width <> H_Large) and
                  (Width <> H_XSmall) and (Width <> H_XLarge) then
                 begin
                   Width := H_Small;
                   Vert := AdjustSc;
                 end { if };
             end { if };
  SmallSc  : Width := H_Small;
  XSmallSc : Width := H_XSmall;
  LargeSc  : Width := H_Large;
  XLargeSc : Width := H_XLarge;
end { cases };

if Width = H_Small then  H_Size := SmallSc
else if Width = H_XSmall then  H_Size := XSmallSc
else if Width = H_Large then  H_Size := LargeSc
else { if Width = H_XLarge then }  H_Size := XLargeSc;

if Horz > SameSc then
  begin
    case H_Size of
      XSmallSc, SmallSc  : Control (Sc_S_Horz);
      XLargeSc, LargeSc  : Control (Sc_L_Horz);
    end { cases };

    Delay;
  end { if };

if (Vert > SameSc) or (Horz > SameSc) then
  begin
    Set_Os_Sc_Size (Height, Width);
    if ReverseSc then  Set_Sc_Color (LightSc);  { Fix Wyse60 error }
    if CrtType <> Unknown then
      begin
      { Make message areas on Wyse 60 normal }  { Fix Wyse60 error }
        S := '*A20*A30';
        S [1] := chr (Pko);     S [5] := chr (Pko);
        W_Str (S);              X := X - 8;
        if not (V_Size in [XSmallSc, XLargeSc]) then
          begin
            S [0] := chr (4);  { delete last 4 characters }
            S [3] := '1';
            W_Str (S);
            X := X - 4;
          end { if };
      end { if };
```

```
      S_Info (InfoStr);
    end { if };
  if I <> Height then
    begin
      Msg_Y := Height;
      for J := Min_Msg to Msg_Tos do begin
        if Msg_Stack [J] .Old_Y = I then
          Msg_Stack [J] .Old_Y := Height;
      end { for };
    end { if };
  Ck_Msg;
end { Set_Sc_Size };
procedure C_Sc { (NoMsg : boolean) };
begin
  Control (Sc_C_Sc);     X := 0;     Y := 0;
  if not NoMsg then  Ck_Msg;
end { C_Sc };
procedure C_Eop { (NoMsg : boolean) };
begin
  Control (Sc_C_Eop);
  if not NoMsg then  Ck_Msg;
end { C_Eop };
procedure C_Eol;
begin
  Control (Sc_C_Eol);
end { C_Eol };
procedure C_Eof { (N : integer) };
var Xx : integer;  Spaces : packed array [0..133] of char;
begin
  Xx := X;
  fillchar (Spaces, N, ' ');
  unitwrite (Console, Spaces, N, , 12);
  X := X + N;  { So that the next SetXy works }
  SetXy (Xx, Y);
end { C_Eof };
procedure Cursor { (Want : OnOff) };
var Have : OnOff;
begin
  if Csr_Stack [Csr_Tos] then  Have := On
  else  Have := Off;
  if Want = Show then  { Force cursor on/off }
    begin
      Want := Have;
      Have := succ (Have);
    end { if }
  else if Want = Pop then
    begin
      if Csr_Tos > 0 then  Csr_Tos := Csr_Tos - 1;
      if Csr_Stack [Csr_Tos] then  Want := On
      else  Want := Off;
    end { if }
  else
    begin
      if Want = Toggle then
        if Csr_Stack [Csr_Tos] then  Want := Off
        else  Want := On;
      if Csr_Tos < Max_Csr then  Csr_Tos := Csr_Tos + 1;
```

```
        Csr_Stack [Csr_Tos] := Want = On;
      end { else };
    if Have <> Want then
      if Want = On then  Control (Sc_S_Csr)
      else  Control (Sc_C_Csr)
{$B CSR+}
    else
      begin
        W_Char ('=');
        X := X - 1;
      end { else }
{$E CSR+}
      ;
  end { Cursor };
  procedure LockKeyboard { (Want : OnOff) };
  var Have : OnOff;
  begin
    if Kbd_Stack [Kbd_Tos] then  Have := On
    else  Have := Off;
    if Want = Show then  { Force lock/unlock }
      begin
        Want := Have;
        Have := succ (Have);
      end { if }
    else if Want = Pop then
      begin
        if Kbd_Tos > 0 then  Kbd_Tos := Kbd_Tos - 1;
        if Kbd_Stack [Kbd_Tos] then  Want := On
        else  Want := Off;
      end { if }
    else
      begin
        if Want = Toggle then
          if Kbd_Stack [Kbd_Tos] then  Want := Off
          else  Want := On;
        if Kbd_Tos < Max_Kbd then  Kbd_Tos := Kbd_Tos + 1;
        Kbd_Stack [Kbd_Tos] := Want = On;
      end { else };
    if Have <> Want then
      if Want = On then  Control (Sc_L_Kbd)
      else  Control (Sc_U_Kbd)
{$B KBD+}
    else
      begin
        W_Char ('=');
        X := X - 1;
      end { else }
{$E KBD+}
      ;
  end { LockKeyboard };
  procedure Control_Atr (Len : integer;  Atr : Sc_Cmds);
  begin
    Len := Len - 1;                SetXy (X + Len, Y);
    Control (Sc_Norm_Atr);         SetXy (X - Len, Y);
    Control (Atr);                 X := X + 1;
  end { Control_Atr };
  procedure S_Rev { (Len : integer) };
  begin
```

```
  Control_Atr (Len, Sc_Rev_Atr);
end { S_Rev };
procedure S_Dim { (Len : integer) };
begin
  Control_Atr (Len, Sc_Dim_Atr);
end { S_Dim };
procedure S_Unln { (Len : integer) };
begin
  Control_Atr (Len, Sc_Unln_Atr);
end { S_UnLn };
procedure S_RevDim { (Len : integer) };
begin
  Control_Atr (Len, Sc_RevDim);
end { S_RevDim };
procedure C_Atr { (Len : integer) };
begin
  Control (Sc_C_Atr);          SetXy (X + Len - 1, Y);
  Control (Sc_C_Atr);          SetXy (X - Len + 2, Y);
end { C_Atr };
procedure W_Str { (S : Str_255) };
begin
  unitwrite (Console, S [1], length (S), , 12);
  X := X + length (S);
end { W_Str };
procedure W_Char { (C : char) };
var Ch : packed array [0..1] of char;
begin
  Ch [0] := C;
  unitwrite (Console, Ch [0], 1, , 12);
  X := X + 1;
end { W_Char };
function Ck_Accept { (Ch : char) : boolean };
begin
  Ck_Accept := ord (Ch) in [Acc, Anf, Apf, Anl, Apl, Ach, Uk, Dk];
end { Ck_Accept };
procedure FlushKeyboard;
begin
  Fcn_Ex := false;
  unitclear (SysTerm);
end { FlushKeyboard };
procedure Next_Char (var C : char);
var Ch : char;  Done : boolean;
  function ReadCh : char;
  var Ch : char;
  begin
    if UseTa then
      begin
        ReadCh := Ta [TaIdx];      TaIdx := TaIdx + 1;
        if TaIdx > TaLast then  UseTa := false;
      end { if }
    else
      begin
        LockKeyboard (Off);        read (keyboard, Ch);
        if eoln (keyboard) then  ReadCh := chr (Eol)
        else  ReadCh := Ch;
```

```
        LockKeyboard (Pop);
      end { else };
  end { ReadCh };
begin { Next_Char }
  repeat
    Done := false;
    if Fcn_Ex then  { Read from function }
      begin
        C := Fcn_Buf [Fcn_I_Ex];      Fcn_I_Ex := Fcn_I_Ex + 1;
        Fcn_Stop := Fcn_Stop + 1;  { for infinite loops }
        if Fcn_I_Ex > Fcn_Keys [Fcn_K_Ex] .Fcn_Last then  Fcn_Ex := false;
        Fcn_Ch := true;  { char came from a function }
      end { if }
    else { read from input }
      begin
        Ch := ReadCh;
        if Ch = chr (Pki) then  { Check for prefixed char }
          begin
            Ch := ReadCh;          Ch := chr (ord (Ch) + P_Bias);
          end { else if };
        C := chr (Translate [ord (Ch)]);
        Fcn_Stop := 0;             Fcn_Ch := false;
      end { else };
    if C = chr (Dbk) then  { Debug key }
      begin
        Debug := not Debug;
        Set_Info;                  Done := OnlyOne;
      end { if }
    else if C = chr (Ftk) then  { Flush toggle key }
      begin
        Flush_On := not Flush_On;
        Set_Info;                  Done := OnlyOne;
      end { else if }
    else  Done := true;
  until Done;
end { Next_Char };
function G_Char { (Chars : CharSet;  Flush, Map : boolean) : char };
label 2, 3;
var Ch : char;  Done, Erased : boolean;  Gx : integer;
  procedure Start_Recording (Key : integer);
  var I, J : integer;
  begin
    with Fcn_Keys [Key] do begin
      if (Fcn_Last <> Null) and (Fcn_First <> Fcn_Date) then  { Krunch }
        if Fcn_Last + 1 = Fcn_I_Rcd then  Fcn_I_Rcd := Fcn_First
        else
          begin
            moveleft (Fcn_Buf [Fcn_Last + 1], Fcn_Buf [Fcn_First],
                                              Fcn_I_Rcd - Fcn_Last - 1);
            J := Fcn_Last - Fcn_First + 1;
            for I := Fcn_F_Key to Fcn_L_Key do begin
              with Fcn_Keys [I] do begin
                if Fcn_First > Fcn_Keys [Key] .Fcn_Last then
                  begin
                    Fcn_First := Fcn_First - J;
                    Fcn_Last := Fcn_Last - J;
                  end { if };
```

```
            end { with };
          end { for };
          Fcn_I_Rcd := Fcn_I_Rcd - J;
        end { else };
      if Fcn_I_Rcd <= Fcn_Max - Fcn_D_Len then
        begin
          Fcn_K_Rcd := Key;
          Fcn_First := Fcn_I_Rcd;
          Fcn_Last := Null;
          Fcn_Rcd := true;
          Set_Info;
        end { if };
    end { with };
  end { Start_Recording };
  procedure Do_Ofk;
  var Block, Msg : integer;  Ch, KeySet : char;  F : FibPtr;
  begin
    In_Ofk := true;              Erased := In_G_Str;
    Ch := G_Char_Answer ([chr (Esc), 'L', 'S'], false, false,
                concat ('[', Fcn_File, ']  L(oad S(ave function keys?'));
    if Ch <> chr (Esc) then
      begin
        Key_Set := G_Char_Answer ([chr (Esc), '0'..'9'], false, false,
                                  'Enter function key set number (0..9)?');
        if KeySet <> chr (Esc) then
          begin
            Block := ord (KeySet) - ord ('0');
            F := Closed;
            if OpenFile (F, Fcn_File, BlkFile, true, Msg) then
              begin
                Block := BlockIo (F, Fcn_Keys, 1, Block, Ch = 'L');
                CloseFile (F, false);
              end { if }
            else  Err_Msg (Msg, Fcn_File);
          end { if };
      end { if };
    In_Ofk := false;
  end { Do_Ofk };
begin { G_Char }
  if Flush and Flush_On then  FlushKeyboard;
  Done := false;     Erased := false;       Gx := X;
  repeat
    Next_Char (Ch);
    if Map then  Ch := Cap (Ch);
    if Ch in Chars then  Done := true
    else if Ch = chr (Rcd) then
      if Fcn_Rcd then  { Stop recording }
        with Fcn_Keys [Fcn_K_Rcd] do begin
          if Fcn_I_Rcd = Fcn_First then
            if Fcn_K_Rcd = Fcn_F_Key + 7 then
              begin
                Fcn_First := Fcn_Date;
                Fcn_Last := Fcn_Max;
              end { if }
            else  Fcn_First := Null
          else  Fcn_Last := Fcn_I_Rcd - 1;
```

```
          Fcn_Rcd := false;       Set_Info;
        end { if with }
      else  { Start recording }
        begin
          Erased := In_G_Str;
          S_Msg (true, false, 'Record what key (<F1>..<F16>)?');
          Cursor (On);
      3:  Next_Char (Ch);
          if (ord (Ch) >= Fcn_F_Key) and (ord (Ch) <= Fcn_L_Key) then
            Start_Recording (ord (Ch))
          else if Ch <> chr (Esc) then
            begin
              Beep;          goto 3;
            end { else if };
          Cursor (Pop);      C_Msg (1);          goto 2;
        end { else }
    else if (ord (Ch) >= Fcn_F_Key) and (ord (Ch) <= Fcn_L_Key) then
      with Fcn_Keys [ord (Ch)] do begin
        if Fcn_Last = Null then  Beep
        else if Fcn_Stop > Fcn_Max * 2 then
          begin
            Fcn_Ex := false;                  Beep;
          end { else if }
        else
          begin
            Fcn_K_Ex := ord (Ch);             Fcn_I_Ex := Fcn_First;
            Fcn_Ex := true;
          end { else };
      end { else if with }
    else if (Ch = chr (Ofk)) and not In_Ofk then  Do_Ofk
    else  Beep;
    if Fcn_Rcd and not Fcn_Ch then
      begin
        Fcn_Buf [Fcn_I_Rcd] := Ch;      Fcn_I_Rcd := Fcn_I_Rcd + 1;
        if Fcn_I_Rcd > Fcn_Max - Fcn_D_Len then
          begin
            Fcn_Keys [Fcn_K_Rcd] .Fcn_Last := Fcn_I_Rcd - 1;
            Fcn_Rcd := false;           Beep;
          end { if };
      end { if if };
  2:
    if Erased then
      begin  W_Str (Gtmp);  SetXy (Gx, Y);  Erased := false;  end { if };
  until Done;
  G_Char := Ch;
end { G_Char };
function G_Chars { (Chars : CharSet;  Map : boolean) : char };
begin
  G_Chars := G_Char (Chars + C_Chars, false, Map);
end { G_Chars };
function G_Cmd { (Chars : CharSet;  var Count : integer) : char };
var Ch : char;  Num : boolean;
begin
  Ch := Cancel;
  repeat
    if Ch = Cancel then
      begin  Num := false;  Count := 0;  end { if }
    else if Ch = Infinity then
```

```
      begin  Num := false;  Count := Null;  end { else if }
    else if (Ch >= '0') and (Ch <= '9') then
      begin
        if Count = Null then  Count := 0;
        Num := true;
        Count := (Count * 10) + ord (Ch) - ord ('0');
      end { else if };
    Ch := Cap (G_Char (Chars + [Cancel, Infinity, '0'..'9'], false, true));
  until Ch in Chars;
  if Num and (Count < 0) then  Count := 1  { overflow error }
  else if not Num and (Count = 0) then  Count := 1;
  G_Cmd := Ch;
end { G_Cmd };
function InputWaiting { : boolean };
var Words : array [0..29] of integer;
begin
  InputWaiting := false;
  if Flush_On then  { ok to check for input characters }
    begin
      unitstatus (SysTerm, Words, 1);
      if Words [0] > 0 then  InputWaiting := true;
    end { if };
end { InputWaiting };
function UserAbort { : boolean };
var Ch : char;
begin
  UserAbort := false;
  if InputWaiting then
    begin
      OnlyOne := true;  { used by Next_Char }
      if G_Char ([chr (0)..chr (127)], false, false) = chr (Esc) then
        UserAbort := true;
      OnlyOne := false;
    end { if };
end { UserAbort };
function G_Str { (var S : Str_255;  Len : integer) : char };
label 2, 3;
var   Xx      : integer;  { Starting location }
      Loc     : integer;  { Current location in Tmp, 1..length (Tmp) }
      Ls      : integer;  { Length of string }
      Ch      : char;
      GaCh    : char;
      First   : boolean;
      Done    : boolean;
      Srch    : boolean;  { Searching for Gck or Gak char }
      InsMode : boolean;
      DelMode : boolean;
      TmpLoc  : integer;
  procedure W_Gtmp;
  var Xx : integer;
  begin
    Xx := X;
    X := Null;  { So SetXy works }
    unitwrite (Console, Gtmp [Loc], Ls - Loc + 1, , 12);
    SetXy (Xx, Y);
  end { W_Gtmp };
```

```
begin  { G_Str }
  Xx := X;
  In_G_Str := Y = Msg_Y;
  Cursor (On);
  GaCh := chr (Nul);
2:
  Gtmp := S;
  if length (Gtmp) > Len then  Ls := Len
  else
    begin
      Ls := length (Gtmp);
      fillchar (Gtmp [Ls + 1], Len - Ls, '_');
    end { else };
  Gtmp [0] := chr (Len);
  W_Str (Gtmp);
  SetXy (Xx, Y);
  Done := false;
  First := Ls > 1;
  Srch := false;
  InsMode := false;
  DelMode := false;
  repeat
    Loc := X - Xx + 1;
    Ch := G_Char (S_Chars, false, false);
    if Ch in L_Chars then
      if Srch then
        begin
          GaCh := Ch;
3: {Gak}  TmpLoc := Loc + 1 + scan (Ls - Loc, = GaCh, Gtmp [Loc + 1]);
          if TmpLoc <= Ls then
            if DelMode then
              begin
                moveleft (Gtmp [TmpLoc], Gtmp [Loc], Ls - TmpLoc + 1);
                TmpLoc := TmpLoc - Loc;
                fillchar (Gtmp [Ls - TmpLoc + 1], TmpLoc, '_');
                W_Gtmp;
                Ls := Ls - TmpLoc;
              end { if }
            else  SetXy (TmpLoc + Xx - 1, Y)
          else  Beep;
          Srch := false;
        end { if }
      else
        begin
          if First then
            begin
              fillchar (Gtmp [2], Ls - 1, '_');
              Gtmp [1] := Ch;
              W_Char (Ch);
              Loc := 2;
              W_Gtmp;
              Ls := 1;
            end { if }
          else if InsMode and (Loc <= Ls) then
            if Ls < Len then
              begin
                Ls := Ls + 1;
                moveright (Gtmp [Loc], Gtmp [Loc + 1], Ls - Loc);
```

```
            Gtmp [Loc] := Ch;
            W_Char (Ch);
            Loc := Loc + 1;
            W_Gtmp;
          end { if }
        else  Beep
      else if Loc <= Len then
        begin
          Gtmp [Loc] := Ch;
          W_Char (Ch);
          if Loc > Ls then  Ls := Ls + 1;
        end { if }
      else  Beep
    end { else }
else if Ch in S_Chars then
  begin
    Srch := false;
    case ord (Ch) of
      Uk,  Dk,
      Acc,
      Anf, Apf,
      Anl, Apl,
      Ach : begin
              while not Done do begin
                if Ls = 0 then  Done := true
                else if Gtmp [Ls] = ' ' then  Ls := Ls - 1
                else  Done := true;
              end { while };
            Gtmp [0] := chr (Ls);
            S := Gtmp;
          end { cases of accept };
      Cfk : First := false;
      Imk : InsMode := not InsMode;
      Dmk : DelMode := not DelMode;
      Esc : Done := true;
      Rtk : begin
              SetXy (Xx, Y);
              goto 2;  { ReTry }
            end { case Rtk };
      Lk  : if Loc > 1 then  SetXy (X - 1, Y)
            else  Beep;
      Rk  : if Loc <= Ls then  SetXy (X + 1, Y)
            else  Beep;
      Tk  : if Loc <= Ls then  SetXy (Xx + Ls, Y)
            else  Beep;
      Btk : if Loc > 1 then  SetXy (Xx, Y)
            else  Beep;
      Ick : if (Loc > Ls) or (Ls = Len) then  Beep
            else
              begin
                moveright (Gtmp [Loc], Gtmp [Loc + 1], Ls - Loc + 1);
                Gtmp [Loc] := ' ';
                Ls := Ls + 1;
                W_Gtmp;
              end { else };
```

```
          Dck : if Loc > Ls then  Beep
                else
                  begin
                    moveleft (Gtmp [Loc + 1], Gtmp [Loc], Ls - Loc);
                    Gtmp [Ls] := '_';
                    W_Gtmp;
                    Ls := Ls - 1;
                  end { else };
          Del : if Loc > Ls then  Beep
                else
                  begin
                    fillchar (Gtmp [Loc], Ls - Loc + 1, '_');
                    W_Gtmp;
                    Ls := Loc - 1;
                  end { else };
          Gck : if Loc > Ls then  Beep
                else  Srch := true;
          Gak : if Loc > Ls then  Beep
                else if GaCh = chr (Nul) then  Beep
                else  goto 3;
          Clk : if Ls = 0 then Beep
                else
                  begin
                    fillchar (Gtmp [1], Ls, '_');
                    Loc := 1;
                    SetXy (Xx, Y);
                    W_Gtmp;
                    Ls := 0;
                  end { else };

        end { cases };
      end { else if }
    else  Beep;
    First := false;
  until Done;
  SetXy (Xx, Y);
  W_Str (S);
  if Len > length (S) then
    C_Eof (Len - length (S));
  G_Str := Ch;
  Cursor (Pop);
  In_G_Str := false;
end { G_Str };
function S_Prompt { (Prompt  : Str_255;
                     Version : Str_7;  Reverse : boolean) : integer };
var Tmp, Xx, Yy : integer;
begin
  Xx := X;
  Yy := Y;
  Tmp := Width - length (Version) - 1;
  if length (Prompt) > Tmp - 3 then
    Prompt [0] := chr (Tmp - 3);
  SetXy (1, Py);
  W_Str (Prompt);
  C_Eol;
  S_Prompt := X + 1;
```

```
  SetXy (Tmp, Y);
  W_Str (Version);
  if Reverse then
    begin
      SetXy (0, Py);
      S_Rev (Width);
    end { if };
  SetXy (Xx, Yy);
end { S_Prompt };
procedure Set_Info;
{ This procedure sets the Debug and Flush indicator in Info.
  It also sets the date.
}
var Tad : TadRec;
begin  {12345678 1 2345678 2 2345}
             {___ 00-00-00 00000 }
  if Flush_On then                          {Flush  Debug}
    if Debug then  Info [7] := '+'          {  On     On }
    else  Info [7] := ' '                   {  On    Off }
  else if Debug then  Info [7] := '-'       { Off     On }
  else  Info [7] := '*';                    { Off    Off }
  if Fcn_Rcd then
    I_into_S (Fcn_K_Rcd - Fcn_F_Key + 1, Info, 8, 2)
  else
    fillchar (Info [8], 2, ' ');
  Get_Sys_Tad (Tad);
  D_into_S (Tad .D, Info, 11);
  S_Info (InfoStr);
end { Set_Info };
procedure S_Info { (S : Str_255) };  { Wyse 60 }
var Xx : integer;
begin
  Xx := X;
  fillchar (Info [20], 11, ' ');
  I_into_S (memavail, Info, 20, 5);
  I_into_S (varavail (''), Info, 26, 5);
  W_Str (Info);
  if length (S) >= InfoWidth then  S [0] := chr (InfoWidth)
  else if length (S) = 0 then
    begin
      S [0] := chr (1);
      S [1] := chr (Eol);
    end { if }
  else if S [length (S)] <> chr (Eol) then
    begin
      S [0] := succ (S [0]);
      S [length (S)] := chr (Eol);
    end { else };
  W_Str (S);
  InfoStr := S;
  X := Xx;  { W_Str changes X }
end { S_Info };
procedure S_Time;
var Xx : integer;  Tad : TadRec;  S : Str_7;
begin
  if CrtType = Wyse60 then
```

```
    begin
      Get_Sys_Tad (Tad);
      if Tad .T <> NullTad .T then
        begin  {1234567}   { Wyse 60 }
          S := '*c80000';
          S [1] := chr (Pko);
          with Tad .T do begin
            I_into_S (Hour, S, 4, 2);
            I_into_S (Min, S, 6, 2);
          end { with };
          Xx := X;
          W_Str (S);
          X := Xx;  { W_Str changes X }
        end { if };
    end { if };
end { S_Time };
procedure S_Msg { (Prompt, Attention : boolean;  S : Str_255) }; { Wyse 60 }
var Xx, Yy : integer;
begin
  if not Clr_Msg then
    begin
      if length (S) > Width - Msg_X - 2 then
        S [0] := chr (Width - Msg_X - 2);
      if Msg_Tos < Max_Msg then  Msg_Tos := Msg_Tos + 1;
      if Prompt then  insert (' ', S, length (S) + 1);
      if length (S) > Msg_StrSz then  S [0] := chr (Msg_StrSz);
      with Msg_Stack [Msg_Tos] do begin
        Old_X := X;
        Old_Y := Y;
        Msg_P := Prompt;
        Msg_A := Attention;
        Msg_S := S;
      end { with };
    end { if };
  if Attention then
    begin
      Init_Msg [3] := '2';    { Reverse blinking message }
      if not Clr_Msg then  Beep;
    end { if }
  else  Init_Msg [3] := '0';  { Reverse message }
  Xx := X;
  Yy := Y;
  SetXy (Msg_X, Msg_Y);
  W_Str (Init_Msg);
  X := X - 2;  { an adjustment for non-display characters in init_Msg }
  W_Str (S);
  C_Eol;
  if not Prompt then  SetXy (Xx, Yy);
end { S_Msg };
procedure C_Msg { (Msgs : integer) };
var I : integer;  Prompt : boolean;
begin
  if Msg_Tos >= Min_Msg then  { There is at least one message }
    begin
      Prompt := false;
```

```
      if (Msg_Tos - Msgs < Min_Msg) or (Msgs = 0) then
        begin
          SetXy (Msg_X, Msg_Y);
          C_Eol;
          Msg_Tos := Min_Msg - 1;
          I := Min_Msg;
        end { if }
      else { if Msg_Tos - Msgs > Min_Msg then }
        begin
          Msg_Tos := Msg_Tos - Msgs;
          I := Msg_Tos + 1;
          with Msg_Stack [Msg_Tos] do begin
            Clr_Msg := true;
            S_Msg (Msg_P, Msg_A, Msg_S);
            Prompt := Msg_P;
            Clr_Msg := false;
          end { with };
        end { else };
      if not Prompt then
        with Msg_Stack [I] do begin
          SetXy (Old_X, Old_Y);
        end { with };
    end { if };
end { C_Msg };

procedure Ck_Msg;
var Xx, Yy : integer;
begin
  if Msg_Tos >= Min_Msg then  { There is at least one message }
    begin
      Xx := X;
      Yy := Y;
      with Msg_Stack [Msg_Tos] do begin
        Clr_Msg := true;
        S_Msg (Msg_P, Msg_A, Msg_S);
        Clr_Msg := false;
      end { with };
      SetXy (Xx, Yy);
    end { if };
end { Ck_Msg };

function G_Char_Answer { (Chars : Char_Set;  Flush : boolean;
                          Attention : boolean;  M : Str_255) : char };
begin
  S_Msg (true, Attention, M);
  Cursor (On);
  G_Char_Answer := G_Char (Chars, Flush, true);
  Cursor (Pop);
  C_Msg (1);
end { G_Char_Answer };

procedure Error { (S : Str_255) };
var Ch : char;
begin
  if length (S) < Width - 12 then
    insert (', <space>', S, length (S) + 1);
  Ch := G_Char_Answer ([' '], true, true, S);
end { Error };

procedure Err_Msg { (Msg : integer;  S : Str_255) };
const Msg_Blk    = 8;  { Messages per block }
      Msg_Blk_M1 = 7;  { minus 1 }
```

```
  var   F          : FibPtr;
        Blk_Rec    : integer;       { Block or record number }
        Ss         : Str_23;
        Block      : array [0..Msg_Blk_M1] of Str_63;
  begin
    if OneErrFile then  Ss := SmsgFile
    else if Msg >= M_UserErr then
      begin
        Ss := UmsgFile;
        Msg := Msg - M_UserErr;
      end { if }
    else  Ss := SmsgFile;
    if OpenFile (F, Ss, BlkFile, true, Blk_Rec) then
      begin
        Blk_Rec := Msg div Msg_Blk;
        if BlockIo (F, Block, 1, Blk_Rec, true) = 1 then  { block found }
          Blk_Rec := Msg mod Msg_Blk
        else  Blk_Rec := Null;
        CloseFile (F, false);
      end { if }
    else  Blk_Rec := Null;
    if Blk_Rec <> Null then
      begin
        if length (S) + length (Block [Blk_Rec]) + 1 >= sizeof (S) then
          S [0] := chr (sizeof (S) - (length (Block [Blk_Rec]) + 1));
        insert (Block [Blk_Rec], S, 1);
      end { if }
    else  { file not found or message empty }
      begin
        if length (S) + 11 >= sizeof (S) then
          S [0] := chr (sizeof (S) - 11);
        insert ('Err #    , ', S, 1);
        I_into_S (Msg, S, 6, 3);
      end { else };
    Error (S);
  end { Err_Msg };
  procedure Set_ErrFiles { (SysErr, UserErr, SoftKeys : Str_23) };
  begin
    if length (SysErr) > 0 then  Smsg_File := SysErr;
    if length (UserErr) > 0 then  Umsg_File := UserErr;
    if length (SoftKeys) > 0 then  Fcn_File := SoftKeys;
    OneErrFile := Smsg_File = Umsg_File;
  end { Set_ErrFiles };
begin { T_Io_U }
  Initialize;
  *** ;
{ Clear info line, except date }
  Info [0] := chr (19);
  fillchar (Info [7], 3, ' ');
  Info [19] := chr (Eol);
  W_Str (Info);
  C_Sc (true);
  Control (Sc_S_Csr);
  Control (Sc_U_Kbd);
end {$O- T_Io_U }.
```

USUS
P.O BOX 1148
LA JOLLA, CA 92038

ADDRESS CORRECTION REQUESTED

USUS NewsLetter
Sept/Oct 1989

Volume 3
William D. Smith, Editor Number 7

NewsLetter Publication Dates

NewsLetter	Due date Code/Forms	Due date Articles	Due date Short stuff
Nov/Dec 89	11/06/89	11/13/89	11/20/89
Jan/Feb 90	01/01/90	01/08/90	01/15/90
Mar/Apr 90	03/05/90	03/12/90	03/19/90
May/Jun 90	05/07/90	05/14/90	05/21/90

Next NewsLetter coming Nov/Dec