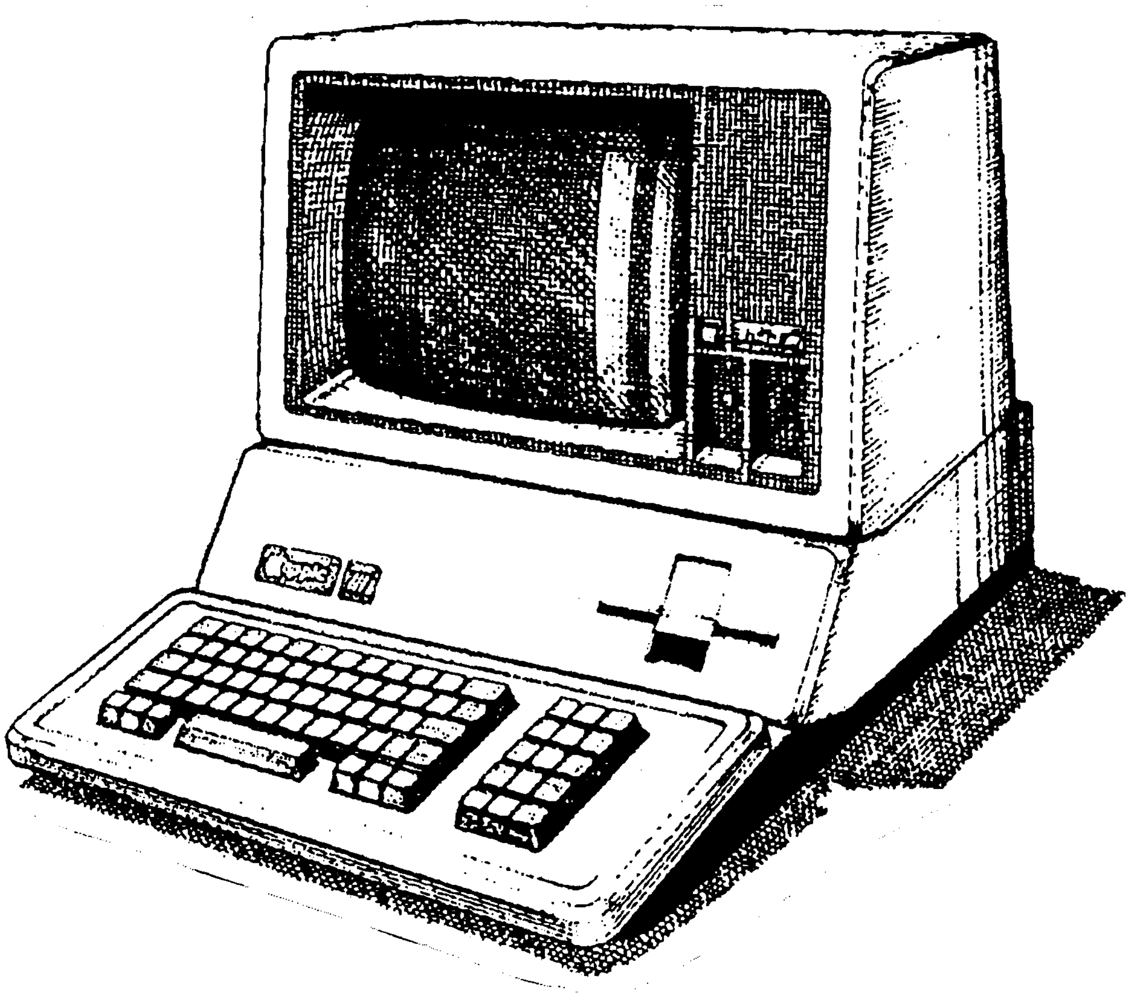
Apple /// Computer Information



DOCUMENT NAME

FONTS AND DOWNLOADING CHARACTERS 1986

# 157

Title: Fonts and downloading characters -
Created by: SYSOP on: 11/22/1988 13:45:08

August 21, 1986 General comments from many people File Length: 7,000 chars.

From: Chris Acreman

Does anybody know how programs like System Utilities and Backup // draw those horizontal lines across the screen? They don't use the GRAFIX driver.
If it is a character, I have not been able to identify it. It is not a series of underscores, across the "bottom" of the line of characters. It looks more like a series of hyphens, across the "middle" of the line, except the hyphens span the full width of the character. If that is an ASCII character, I haven't found it.
Thanks for any help you may offer.

Chris

Chris;
if you have a program for font editting like Fontwriter or Draw On ///, you can load the entire font in to look at it. It is possible that these are characters that are assigned to Ascii 0-32, i.e. control characters. Capn Magneto works off this principle.

Weber Baker

Chris: I believe both programs are using a custom font. I have the source code here in ///'s Company for a Pascal program that will extract the font from any boot disk for you and save it in a separate file, from which you can then install it on any boot disk of your choice by using the SCP.

....Ed Gooding

Ed, Is it possible to change fonts from within a Pascal Program? I.e. - can you modify specific characters or completely replace the entire font setup.

I want my application to use a custom font set and then set it back to normal when the program terminates.

Thanks. Harry Baya

Harry:

See Rick Sidwell's response to Chris below. I was wrong about the font, it is done differently. It appears to be done exactly the way you want to replace your font characters and then restore them. I suggest that you get in touch with Rick for more details; this was news to me, also. Thanks, Rick.

Ed

Chris/Ed/Harry:

The underscore-like characters are made with custom characters, but they are not in the font on the system utilities disk--otherwise, how could you run system utilities from your Profile as a normal Pascal program. The .CONSOLE driver has a control call for downloading a few characters. System Utilities redefines a few characters for its `graphics', and restores the original characters when you exit. If you have Pascal, try running it from the Pascal system (i.e., not as the SYSTEM.STARTUP program, then using Control-\ to break out of the program. The characters will be intact. (Sorry, I can't remember where they are--try the highest control characters).

Rick Sidwell

Chris -

Those lines you're asking about don't seem to be any special characters in the character font--I've used the FONT CAPTURE program in DL7 to extract the character set from a SOS boot disk, then used the ENCD3.BAS program to change the filetype from PASCAL DATA to FONT, and finally used ON THREE's FONTDEMO program (in BASIC) to look at the complete character set: Nothing there that would satisfy your question. I know that's no help in identifying how it IS done . . .

- Bart Cable

Chris/Ed/Harry/Bart/All,

The characters which do the borders in System Utilities and similar packages are special characters which are downloaded AT THE TIME THE APPLICATION IS STARTED. They appear to be within the program code itself, and are not a separate character set. For System Utilities, they exist in the ASCII values from 16 to 31. To do this yourself, you will need to perform the following:

1. Use a font (character set editor) to create your own character set which contains special charcters for vertical and horizontal bars, and curved corners. You can also do left and right arrows, etc. Save the character set to a file name to be loaded later by a Pascal program.

2. Set up your program to read the font file from step 1 above and send it to the console driver. See the Pascal Programmer's Manual Volume 1 (pg 211 - UNITSTATUS) and the Standard Device Drivers Manual (pg 70 - Control code 16 [Download Character Set]).

3. Use the UNITWRITE procedure (see Pascal Programmer's Manual Volume 1 - page 207) to send the character values to the screen to create the "box" where you want it or to send any other of the special characters to the screen. As I remember, you have to add 128 to the value of each such character so that it will not be interpreted by the console driver as a control character. For example, if ASCII 16 is a horizontal bar you have to send the ASCII value 144 to the console to get the character, rather than the control effect of setting the color as specified in the Standard Device Drivers Manual.

As an alternative to step 1 you can define bit patterns for the characters you wish to modify in your Pascal program and use the Control Code 17 (Load Partial Character Set) to create the special characters.

If you have Daryl Anderson's Ascii Chart module you can view the characters used in System Utilities by activating it within System Utilities. If you have Desktop Manager, ask Bob Consorti for a copy of his Ascii Chart module that I modified to show the control characters, and implement it within System Utilities.

Hope this helps.

Best Regards,
Milt Johnson

Toll:

I'll tell you how to do it, but I won't have time for a couple of weeks to come up with some code to do it (I'm going on vacation). You use the CONSOLE driver control code 16 (for the entire character set) or 17 (for just a few characters) to change the character set to what you want. To restore it, you use the same codes! The trick is to save the original characters before changing it in the first place. The current character set is stored at locations $C00-$FFF. You will need an assembly language procedure to access these locations from Pascal.

Rick Sidwell

Dear Rick,
Please do enjoy your vacation, but when you return, I WILL bug you about this, gently of course. I will very much appreciate your help since I know boobkis about assembly language. Thanks for your tips and I'll look forward to that "tutorial". -- Chris

Rick, I too would greatly appreciate some sample code to modify fonts from within a program. It could be as short as just the lines to cause the actual change - without a good demo program - or whatever you want.

Have a good vacation and I and Chris will remind you, gently, when you get back.

Thanks, Harry

To All:

Just about everything you want to know about loading fonts:

"The Changing Character(s) of the Apple ///" by John Jeppson
Softalk, April 1982. Pages 135-142

The program in the article is in MAUG's DL7 under the name "JEP"

briefly:

```
10 INVOKE "download.inv"
20 DIM a%(512):array$="a%"
30 INPUT"What font do you want to use? ";a$
40 expr$=CHR$(34)+a$+CHR$(34)
50 PERFORM getfont(@expr$,@array$)
60 PERFORM loadfont(@array$)
70 END
```

--Erik Olbrys

09/12/86 13:13:36

The following was contributed by Rick Sidwell, and answers the questions posed above about manipulating characters sets from within an application program:

DOWNLOADING APPLE /// CHARACTER SETS

One of the powerful features of the Apple /// is the ability to change the character set used to display characters on the screen easily from a program. A program can define its own characters to perform special effects such as drawing graphics on the text screen. A good example is the System Utilities program which uses custom characters to draw boxes around things and display arrows as they appear on the keyboard to help the user. However, any program which changes the system character set should be careful to restore it before exiting so that other programs can use the normal character set. This isn't very difficult to do, but it requires the use of assembly language, and is thus a bit tricky.

For purposes of this discussion, let's use Pascal. Similar techniques can be used with other languages. To begin with, we need to have a TYPE for character sets:

```
        Type Charset = Packed Array [0..127, 0..7] of 0..255;
```

A character set consists of 128 characters (numbered 0 through 127), each consisting of 8 rows. Next, we need a procedure to download character sets. Referring to the Standard Device Drivers manual, pages 70 and 169-171, the following procedure will do the trick:

```
        Procedure LoadCharset(C:Charset);
        Var RequestCode: Packed Record
                           Channel: 0..1;
                           Stat_or_Ctrl: 0..1;
                           Request_Num: 0..255;
                           Reserved: 0..63;
                         End;
        Begin { LoadCharset }
        RequestCode.Channel := 0;
        RequestCode.Reserved := 0;
        RequestCode.Stat_or_Ctrl := 1;
        RequestCode.Request_Num := 16;
        UnitStatus(1,C,RequestCode);
        End { LoadCharset };
```

This just performs a UnitStatus to the .CONSOLE driver with a request code to download a character set. So far, so good; now for the tricky part: restoring the system character set. To do this, we need to copy the system character set before we download our own; restoring it when we are done is as easy as another call to LoadCharset. The current character set is stored in system memory at locations $C00-$FFF. The .CONSOLE driver stores the new character set here as well as loading it into the character generator so that the .GRAFIX driver can use it for drawing characters onto the graphics screen. Note that anyone can read this character set, but only the .CONSOLE driver should modify it so that it remains consistent with the character set displayed on the text screen. Copying data from system memory to Pascal memory can be done only from assembly language. For the convenience of programmers not proficient in assembly language in the Apple ///, here is a complete assembly language procedure which copies the character set.

```
; Assembly procedure to copy the system character set to a user variable

; Pascal interface:  Procedure SysCharset(Var C:Charset);

; Some standard macros
        .MACRO  POP
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM

        .MACRO  PUSH
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM

; Some zero page temporaries
Return          .EQU    0E0             ;To save return address
Ptr             .EQU    0E2             ;Pointer to user's variable
```

```
SysSet          .EQU    0E4             ;Pointer to system charset

        .proc   SysCharset,1

        POP     Return                  ;Save return address
        POP     Ptr                     ;Get location to put charset
        LDA     #00                     ;Set up pointer to system charset
        STA     SysSet
        LDA     #0C
        STA     SysSet+1
        LDA     SysSet+1601             ;Save old X-byte
        PHA
        LDA     #8F                     ;System charset is in system bank
        STA     SysSet+1601

        LDY     #0
NxtChr  LDA     (SysSet),Y              ;Copy a character
        STA     (Ptr),Y
        INY                             ;Do next one
        BNE     NxtChr
        INC     Ptr+1
        INC     SysSet+1
        LDA     SysSet+1                ;See if done (if we reached $1000)
        CMP     #10
        BCC     NxtChr

        PLA                             ;Restore X-byte for Pascal
        STA     SysSet+1601
        PUSH    Return
        RTS

        .END
```

The code is straightforward for those familiar with assembly code; the only tricky part is saving the X-byte of the variable we use to access system memory (SysSet) and restoring it before returning so that Pascal will not get confused. To use it, copy it into a file, assemble it, and then link it into your program. It defines a procedure called SysCharset which copies the current character set into a Pascal variable. Here is an example program to demonstrate its use:

```
Program TestCharset;

Type Charset = Packed Array [0..127, 0..7] of 0..255;

Var SysSet: Charset;
    C: Char;
    S: String;
    F: File of Charset;

Procedure SysCharset(Var C:Charset); External;
  { The assembly language program to get the system character set }

Procedure LoadCharset(C:Charset);
{ Loads the character set C into the character generator }
Var RequestCode: Packed Record
                   Channel: 0..1;
                   Stat_or_Ctrl: 0..1;
                   Request_Num: 0..255;
                   Reserved: 0..63;
                 End;
Begin { LoadCharset }
RequestCode.Channel := 0;
RequestCode.Reserved := 0;
RequestCode.Stat_or_Ctrl := 1;
RequestCode.Request_Num := 16;
UnitStatus(1,C,RequestCode);
End { LoadCharset };
```

```
Begin { Main program }
{ First, save the system character set in SysSet }
SysCharset(SysSet);

{ Ask the user for a file with a new character set }
Write('Character set to load: ');
Readln(S);
Reset(F,S);

{ Load the user's character set and close the file }
LoadCharset(F^);
Close(F);

{ Put some characters on the screen to show off the new character set }
For C:=' ' to '~'
Do
   Write(C);
Writeln;

{ Wait until the user is ready to exit }
Write('Press return to exit. ');
Readln;

{ Restore the old character set before exiting }
LoadCharset(SysSet);
End.
```

EXERCISES

1. It is often not necessary to download a complete character set. The System Utilities program, for example, downloads only a few characters so that it can draw boxes and arrows. Explain how to do this.

2. Write an invokable module for Business Basic so that programmers can write Basic programs which use custom fonts but restore the system font before exiting.

10/03/1986 01:31:18
I find it very interesting that this discussion has been going on here... I was just about to upload a related file, so I will do it here. System Utilities uses to handle its windows, horizontal lines, etc. characters established by the Dirstuff and Prims2 units, wwhich have been "Krunched" into the SYSTEM.STARTUP codefile using a Library Kruncher program, which strips away the interfaces and binds the units together with the codefile. The Dirstuff unit is well documented in the "Pascal Programmer's Toolkit Manual" available from ATUNC, and that documentation also covers some of the Prims2 routines, although it never specifically mentions them as Prims2. The only documentation for Prims2 is the comments in the INTERFACE section, which I have extracted using LIBMAP.CODE (which I also used to examine System Utilities... got the unit names, but no interface there) and provide here for your perusal. Note that these routines do all those things WITHOUT requiring the .GRAFIX driver... they are entirely self contained, and I am sure with a little experimentation, one or more ingenious hacker could figure a way to load ANY character set using them. As a side benefit, of course, one also has the handy file selection and keyboard I/O capabilities provided by Dirstuff available.

Below you will find the file, as I prepared it for upload:

Tom Betz

Friends and fellow ///ers!

```

   As promised so long ago, here's the INTERFACE section to to the PRIMS2
   Library Unit, as provided by LIBMAP.CODE; some interesting routines named
   here, as well as the major control characters defined as constants.
   Anybody have any ideas what they are?  I have heard them mentioned in
   terms of Kernigan and Ritchie.. don't know enough about C to make a
   meaningful comparison.  I'd be interested in comments... if you like, I'll
   run Codefile Transmitter over the Unit and upload it... what do you think?

    Tom Betz



Segment #38:
System version = A3/2.0, code type is 6502
PRIMS2    library unit (LINKED INTRINSIC)


      {$p----------------------------------------------------*
       | PRIMS2 - String Version of Primitives - Updated 08/28/83 |
       *----------------------------------------------------*}

      USES sosio, chainstuff;

      CONST {[j=13/40]}
         max_open     = 12;          {max number of open files allowed}

         ioread       = 0;           {read mode for a file}
         iowrite      = 1;           {write mode for a file}

         f_avail      = 0;           {file types}
         f_text       = 1;           {}
         f_ascii      = 2;           {}
         f_char       = 3;           {}
         f_code       = 4;           {}

         maxstr       = 132;         {max length of a "strng"}

         {Special font characters defined by define_new_font}
         err_u_arrow  = 134;         {up arrow used in error msg}
         err_d_arrow  = 135;         {down arrow used in error msg}
         err_l_arrow  = 136;         {left arrow used in error msg}
         err_r_arrow  = 137;         {right arrow used in error msg}
         left_side    = 138;         {left side bar/dash intersection}
         right_side   = 139;         {right side bar/dash intersection}
         top_left     = 140;         {top left corner for window frames}
         top_right    = 141;         {top right corner for window frames}
         bot_left     = 142;         {bottom left corner for window frames}
         bot_right    = 143;         {bottom right corner for window frames}
         bar          = 144;         {vertical bars on window frames}
         dash         = 145;         {1st char of the right arrow}
         right_arrow  = 146;         {2nd right arrow char}
         up_arrow     = 147;         {up arrow char for "more above"}
         down_arrow   = 148;         {down arrow char for "more below"}
         up1          = 149;         {1st third of "more" for "more above"}
         up2          = 150;         {2nd third of "more" for "more above"}
         up3          = 151;         {3rd third of "more" for "more above"}
         down1        = 152;         {1st third of "more" for "more below"}
         down2        = 153;         {2nd third of "more" for "more below"}
         down3        = 154;         {3rd third of "more" for "more below"}

      TYPE                           {$p}
         filedesc     = integer;     {file descriptors - NOT SOS ref num}

         strng        = string[140]; {string format}

         bufr_ptr     = ^io_buffer;  {I/O buffer pointers}

```

"_101.PICT" 225 KB 2001-08-29 dpi: 300h x 300v pix: 2368h x 3113v

```
        io_buffer    = PACKED ARRAY [0..1025] OF char; {I/O buffers with 2
                                                         nulls at 1024-5}

        f_entry_bufr = PACKED ARRAY [0..38] OF 0..255; {file entry buffers}

        fct          =
                       RECORD          {File Control Table}
                          ref_nbr: integer;
                          ft: f_avail..f_code;
                          ct: (c_console, c_printer, c_silent, c_other);
                          mode: ioread..iowrite;
                          filename: strng;
                          curr_line: strng;
                          line_len: integer;
                          line_cp: integer;
                          pvt_bufr: boolean;
                          buffer: bufr_ptr;
                          bufr_cp: 0..1025;
                          bufr_size: 0..1024;
                          end_of_file: boolean;
                       END;

        io_blk_ptr   = ^io_blk;        {pointer to standard I/O blocks}

        io_blk       = PACKED ARRAY [0..511] OF 0..255; {a standard I/O block}

        cons_buffer  = PACKED ARRAY [0..2048] OF char; {big buffer for screen}

        setofchar    = SET OF char;    {a parameter type to some Primitives}

     VAR                               {[j=15/40]} {$p}
        nargs:          integer;       {total number of arguments}
        options:        setofchar;     {command options}
        ret_to_shell:   boolean;       {true==>return to shell}
        shell_name:     string;        {shell name (default *shell/shell.code)}

        stdin:          filedesc;      {standard input file descriptor}
        stdout:         filedesc;      {standard output file descriptor}

        console:        integer;       {.CONSOLE SOS I/O ref nbr}
        cons_d_num:     integer;       {device number for console}

        curr_prefix:    string;        {current Pascal system filename prefix}
        exec_prefix:    string;        {executing program prefix}

        must_prefix:    strng;         {"$" prefix for mustopen and mustcreate}
        must_suffix:    strng;         {"$" suffix for mustopen and mustcreate}
        must_x, must_y: integer;       {screen x/y set by mustgetarg which are
                                        used by mustopen and mustcreate}

        sel_menu_x, sel_menu_y: integer; {selector menu x/y coordinates}
        sel_menu_len:   integer;      {nbr of names shown in menu at any time}

        lib_sw:         boolean;       {true==>treat '*' as *system.library.}
        err_bell_sw:    boolean;       {true==>let write_error ring the bell}
        dot_text_code:  boolean;       {true==>add .text/.code if required}
        codef_setup:    boolean;       {true==>setting up a codefile}

        valid_chars:    setofchars;    {valid read_keyboard chars}

        {Universal Character Constants}
        endstr:         char;          {$00 end of string char}
        endfile:        char;          {$03 end of file char}
        backspace:      char;          {$08 backspace char}
        tab:            char;          {$09 tab character}
        newline:        char;          {$0D new line character}
        escape:         char;          {'\' special char trigger}
```

```
              {Ascii codes $00 to $32}      {[j=0/0,f-]}
        a_nul { 00 } , a_soh { 01 } , a_stx { 02 } , a_etx { 03 } , a_eot { 04 } ,
        a_enq { 05 } , a_ack { 06 } , a_bel { 07 } , a_bsp { 08 } , a_ht  { 09 } ,
        a_lf  { 10 } , a_vt  { 11 } , a_ff  { 12 } , a_cr  { 13 } , a_so  { 14 } ,
        a_si  { 15 } , a_dle { 16 } , a_dc1 { 17 } , a_dc2 { 18 } , a_dc3 { 19 } ,
        a_dc4 { 20 } , a_nak { 21 } , a_syn { 22 } , a_etb { 23 } , a_can { 24 } ,
        a_em  { 25 } , a_sub { 26 } , a_esc { 27 } , a_fs  { 28 } , a_gs  { 29 } ,
        a_rs  { 30 } , a_us  { 31 } , a_sp  { 32 } : char;

            have_error:    boolean;        {true==>and error was detected}

             cons_bufr:     ^cons_buffer;  {ptr to big console buffer}
             cons_len:      integer;       {number of bytes in "cons_bufr"}

             open_tbl:      ARRAY [1..max_open] OF fct; {File Control Tables}
                                    {$p-----------*
                                     | Primitives |
                                     *------------*}

          PROCEDURE str(n: integer; VAR s: string);
       [NOTE: Here's ^ the very devil that causes my incompatibility problems!]
          PROCEDURE trim_blanks(leading: boolean; VAR s: strng; trailing: boolean);

          PROCEDURE get_td(VAR s: string);

          FUNCTION  open_directory(dir_name: strng; bufr: io_blk_ptr): integer;

          PROCEDURE close_directory;

          FUNCTION  next_dir_entry(VAR file_entry: f_entry_bufr): boolean;

          FUNCTION  dev_or_vol(name: strng): boolean;

          PROCEDURE setup_filename(VAR name: strng; VAR ftype: integer);

          FUNCTION  openf(VAR name: strng; mode: integer; bufr: bufr_ptr):
filedesc;

          FUNCTION  createf(VAR name: strng; mode: integer; bufr: bufr_ptr):
filedesc;

          PROCEDURE next_page(fd: filedesc; bufr: bufr_ptr; VAR size: integer;
                        {Oolean);

          PROCEDURE flush_buffer(fd: filedesc);

          PROCEDURE put_out_line(fd: filedesc; VAR line: strng; len: integer);

          FUNCTION  getcf(VAR c: char; fd: filedesc): char;

          PROCEDURE putcf(c: char; fd: filedesc);

          FUNCTION  getline(VAR s: strng; fd: filedesc): boolean;

          PROCEDURE closef(fd: filedesc);

          PROCEDURE remove(name: strng);

          FUNCTION  set_echo(on_or_off: boolean; fd: filedesc): boolean;

          FUNCTION  have_input(fd: filedesc): boolean;

          FUNCTION  getc(VAR c: char): char;

          PROCEDURE putc(c: char);

          PROCEDURE putstr(s: strng; fd: filedesc);

```

```
    PROCEDURE putline(VAR s: strng; fd: filedesc);

    PROCEDURE get_sos_error(sos_rc: integer; VAR msg: strng);

    PROCEDURE p_error(msg: strng);

    PROCEDURE p_message(s: strng; fd: filedesc);

    FUNCTION  open_odometer(name: string; x, y: integer): integer;

    PROCEDURE display_odometer(od: integer);

    PROCEDURE set_odometer(n, od: integer);

    PROCEDURE close_odometer(od: integer);

    FUNCTION  interrupted: boolean;

    PROCEDURE read_keyboard(VAR line: strng; delimiters: setofchar;
                            VAR delim: char; curs_at_eol: boolean);

    PROCEDURE getxy(VAR x, y: integer);

    PROCEDURE get_viewport_limit(VAR x, y: integer);

    FUNCTION  define_new_font: boolean;

    PROCEDURE restore_orig_font;

    PROCEDURE write_screen;

    PROCEDURE put_out_str(s: strng);

    PROCEDURE put_out_c(c: integer);

    PROCEDURE point(x, y: integer);

    PROCEDURE set_window(x1, y1, x2, y2: integer);

    PROCEDURE frame_window(x1, y1, x2, y2: integer);

    PROCEDURE write_error(line1, line2, line3, line4: strng;
                          responses: setofchar; VAR resp: char);

    PROCEDURE clear_error;

    FUNCTION  getarg(n: integer; VAR s: strng): boolean;

    FUNCTION  get_arg_value(n: integer): integer;

    FUNCTION  expand_filename(prompt: strng; VAR x, y: integer;
                              VAR fname, prefix, suffix: strng): boolean;

    PROCEDURE must_get_arg(n: integer; filename: boolean; prompt: strng;
                           VAR arg: strng);

    FUNCTION  mustopen(VAR name: strng; mode: integer; bufr: bufr_ptr;
                       prompt: strng): filedesc;

    FUNCTION  mustcreate(VAR name: strng; mode: integer; bufr: bufr_ptr;
                         prompt: strng): filedesc;

    FUNCTION  prompt_for_filename(prompt: strng; VAR x, y: integer;
                                  VAR fname: strng;
                                  curs_at_eol: boolean): boolean;

    FUNCTION  search_directory(VAR name: strng; VAR selections: bytestream;
                               VAR max_selected, rc: integer): boolean;
```

"_104.PICT" 192 KB 2001-08-29 dpi: 300h x 300v pix: 2368h x 3113v

```

       PROCEDURE get_options(valid_options: setofchar; options_file: strng;
                             VAR options: setofchar);

       PROCEDURE init_cmd(valid_options: setofchar; options_file: strng;
                          get_args: boolean; in_bufr, out_bufr: bufr_ptr);

       PROCEDURE end_cmd;




      ----------------------------------------------------------------------

Segment #39:
System version = A3/2.0, code type is P-Code (least sig. 1st)
PRIMS2    data segment

      ----------------------------------------------------------------------

      -------
You will note that for windowing, horiz lines, etc, you need nothing more than
this....   This library unit could easily be the target of many hours of
interesting experimentation!  I recommend getting the Toolkit and docs from
ATUNC (sysop's note: Apple Three Users of Northern California - see the
III.User.Groups/ directory) for all you Pascal hackers out there...

  Regards,
  Tom Betz

```