Publication 84-0850-203
Issued March 1992

TABLET.COM™

Programming Reference

Summagraphics Corporation
Sixty Silvermine Road, Seymour CT 06483-3907 U.S.A. Telephone: (203) 881-5400 Fax: 203-881-5367

Please read this license carefully before using the software.

By using the software, you are agreeing to be bound by the terms of this license. If you do not agree to the terms of this license, promptly return the unused software to the place where you obtained it and your money will be refunded.

License - The software accompanying this License, whether on disk, in read only memory, or on any other media (the "Software") and related documentation are licensed to you by Summagraphics Corporation. You own the disk on which the Software is recorded but Summagraphics Corporation retains title to the Software and related documentation. This License allows you to use the Software on a single computer and make one copy of the Software in machine readable form for backup purposes only. You must reproduce on such copy the Summagraphics Corporation copyright notice and any other proprietary legends that were on the original copy of the Software. You may also transfer all your license rights in the Software to another party, provided the other party reads and agrees to accept the terms and conditions of this License.

Restrictions - The Software contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Software to a human-perceivable form. You may not modify, network, rent, lease, loan, distribute or create derivative works based upon the Software in whole or in part.

Termination - This license is effective until terminated. You may terminate this License at any time by destroying the Software and related documentation and all copies thereof. This License will terminate immediately without notice from Summagraphics Corporation if you fail to comply with any provisions of this License. Upon termination you must destroy the Software and related documentation and all copies thereof.

Limited Warranty On Media - Summagraphics Corporation warrants the media on which the Software is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of purchase as evidenced by a copy of the receipt. Summagraphics Corporation's entire liability and your exclusive remedy will be replacement of the media not meeting Summagraphics Corporation's limited warranty and which is returned to Summagraphics Corporation or a Summagraphics Corporation authorized representative with a copy of the receipt. Summagraphics Corporation will have no responsibility to replace media damaged by accident, abuse or misapplication. Any implied warranties on the media, including the implied warranties of merchantability and fitness for a particular purpose, are limited in duration to ninety (90) days from the date of delivery. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Disclaimer of Warranty - You expressly acknowledge and agree that use of the Software is at your sole risk. The Software and related documentation are provided "AS IS" and without warranty of any kind and Summagraphics Corporation expressly disclaims all warranties, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Summagraphics Corporation does not warrant that the functions contained in the software will meet your requirements, or that the operation of the software will be uninterrupted or error-free, or that defects in the software will be corrected. Furthermore, Summagraphics Corporation does not warrant or make any representations regarding the use or the results of the use of the software or related documentation in terms of their correctness, accuracy, reliability, or otherwise. No oral or written information or advice given by Summagraphics Corporation or a Summagraphics Corporation authorized representative shall create a warranty or in any way increase the scope of this warranty. Should the software prove defective, you (and not Summagraphics Corporation or a Summagraphics Corporation authorized representative) assume the entire cost of all necessary servicing, repair or correction. Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Limitation of Liability - Under no circumstances, including negligence, shall Summagraphics Corporation be liable for any incidental, special, or consequential damages that result from the use or inability to use the software or related documentation, even if Summagraphics Corporation or a Summagraphics Corporation authorized representative has been advised of the possibility of such damages. Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you. In no event shall Summagraphics Corporation's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Software.

Controlling Law and Severability - This license shall be governed by and construed in accordance with the laws of the United States and the State of Connecticut, as applied to agreements entered into and to be performed entirely within Connecticut between Connecticut residents. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.

Complete Agreement - This License constitutes the entire agreement between the parties with respect to the use of the Software and related documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Summagraphics Corporation.

Export Law Assurances - You agree and certify that neither the Software nor any other technical data received from Summagraphics Corporation, nor the direct product thereof, will be exported outside the United States except as authorized and as permitted by the laws and regulations of the United States.

Government End Users - If you are acquiring the Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees:
(i) if the Software is supplied to the Department of Defense (DoD), the Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and
(ii) if the Software is supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Software and its documentation will be defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

Disclaimer of Warranty on Licensors Software
"Summagraphics Corporation's Licensor(s) makes no warranties, express or implied, including without limitation the implied warranties or merchantability and fitness for a particular purpose, regarding the Software. Summagraphics Corporation's Licensor(s) does not warrant, guarantee or make any representations regarding the use or the results of the use of the software in terms of its correctness, acuracy, reliability, currentness or otherwise. The entire risk as to the results and performance of the software is assumed by you. The exclusion of implied warranties is not permitted by some states. The above exclusion may not apply to you."

"In no event will Summagraphics Corporation's Licensor(s), and their directors, officers, employees or agents collectively Summagraphics Corporation's Licensor be liable to you for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, and the like) arising out of the use or inability to use the software even if Summagraphics Corporation's Licensor has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. Summagraphics Corporation's Licensor's liability to you for actual damages from any cause whatsoever, and regardless of the form of the action (whether in contract, tort (including negligence), product liability or otherwise), will be limited to the amount paid by you for the software."

Table of Contents

Table Of Contents

Chapter 1
Introduction

troduction

1 About this Book

This manual is a reference for programmers developing applications programs that utilize the Summagraphics tablet driver TABLET.COM™. It is written for both novice and advanced programmers. Software features described in the manual are labelled as "Basic" and "Advanced" for your convenience.

Note: The driver is a complex extension to DOS which interacts directly with hardware registers, the display card, and the BIOS. In the PC emulation environment these interactions can become extremely complicated. Complex and subtle software faults can be created by careless use of the advanced features of the driver. We have included lots of detail for the advanced programmer, with the caution that if you don't understand the potential problems associated with a given feature you probably shouldn't use it.

2 About TABLET.COM

TABLET.COM is a driver designed for absolute pointing devices such as digitizing tablets. It is a memory resident program that provides a mouse driver emulation and additional functions that are unique to absolute pointing devices. The mouse component of TABLET.COM is functionally equivalent to the industry-standard Microsoft® mouse driver, which means that TABLET.COM can be used by any applications that use this mouse driver. The additional functions provided by TABLET.COM take advantage of the digitizer's ability to specify an exact position in two-dimensional space. TABLET.SYS provides identical functionality but is intended for loading at startup time from a CONFIG.SYS file.

Perhaps the most important feature that TABLET.COM has to offer is the ability to provide a mouse emulation that scales a rectangular region on the digitizing surface to a mouse cursor position on the video monitor. This allows, for example, the tracing of a "hard copy" drawing into a CAD or paint program. Furthermore, this rectangular region can be defined to be anywhere on the digitizing surface.

.3 Installing and Using TABLET.COM

Command Syntax

TABLET [com port][stylus/cursor][tracking][other options]

where [com port], [stylus/cursor], [tracking], and [other options] are replaced with the options listed below. If you type only TABLET, the tablet defaults to those options marked with an asterisk.

Com Port Options

/1	Com 1 *
/2	Com 2
/3	Com 3, only on IBM PS/2 systems
/4	Com 4, only on IBM PS/2 systems
/BAxxxx	selects serial port with a base address of xxxx (address must be in hexadecimal)
/Ix	selects serial port with IRQ line number x (2-7)

1.3 Installing and Using TABLET.COM (cont.)

Stylus/Cursor Options

/CP two-button stylus *
/C3 three-button stylus
/C4 four-button cursor

/CM#### allows a user-defined mapping of the cursor/stylus buttons. The /CM argument is followed by 0-4 digits that specify which cursor/stylus switches are assigned to standard mouse buttons. Each digit represents the value of a cursor/stylus switch, and its position represents the mouse button that it is assigned to. The first digit after the /CM argument corresponds to the left mouse button, the second digit to the right mouse button, the third digit to the middle mouse button, and the fourth digit corresponds to an emulation of both the left and right mouse buttons being pressed simultaneously. If the /CM argument has no numerical arguments following it, then the default button mapping for the current cursor/stylus is assumed.

Example

TABLET /CM2134

The above example shows cursor/stylus switch #2 being mapped to the left mouse button, cursor/stylus switch # 1 being mapped to the right mouse button, cursor/stylus switch #3 being mapped to the middle mouse button, and cursor/stylus switch #4 being mapped to both the right and left mouse buttons.

Tracking Options

The following options will work only if the application is set up to use a Microsoft Mouse (MOUSE.COM).

Applications that are compatible specifically with TABLET.COM should configure tracking automatically or allow you to configure tracking through the application's set-up utility. The use of the following options will be overridden by the application.

Relative Mode /R *
Relative mode is the default setting. There are four relative mode options:

/S## Sets both horizontal and vertical sensitivity - - Range=0-99, default=50

/H## Sets only the horizontal sensitivity - - Range=0-99, default=50

/V## Sets only the vertical sensitivity - - Range=0-99, default=50

/M## This option is the ballistic gain profile. It controls acceleration of the screen cursor. TABLET.COM comes with four built-in ballistic gain profiles: Slow, Moderate, Fast, and Unaccelerated. Specify 1,2,3, or 4 respectively.

1.3 Installing and Using TABLET.COM (cont.)

Tracking Options (cont.)

Absolute Mode /A
Absolute mode maps the tablet's entire active area to the computer screen. Relative mode options (/S, /H, /V, /M) should not be used in Absolute mode.

Note: Applications accept input from MOUSE.COM as either absolute screen coordinates, or relative mouse units, also referred to as "mickeys". Absolute mode works only with those applications that accept input as absolute screen coordinates. To verify that absolute mode is working, position the screen pointer, then lift the stylus/cursor out of the tablet's active area and place it at a different location on the tablet. The screen cursor should snap to a new location. If it does not, then the application works only in relative mode and the absolute mode option will have no effect.

Other Options

/O# This option overrides an application that changes the mouse cursor for text modes by "locking" the mouse cursor to the default text cursor. To enable locked cursor type /O1; to disable locked cursor type /O0.

/N### This option specifies how many screen updates to skip before actually plotting the cursor. This feature is useful for laptop computers and other systems with slower LCD displays. A small value is recommended because driver performance can be affected. ### is a number between 0-255.

Notes

- When selecting a serial port other than 1,2,3, or 4, the /BA and /I options must be used together.
- When using the /BA option, all four digits must be specified (Example: for Com 1 - /BA03F8)
- Use the /BA and /I options only when a non-standard serial address is used in the computer.

Disabling The Driver

To disable TABLET.COM type:

```
TABLET /OFF
```

1.4 Basic Driver Function

TABLET.COM allows for four modes of interaction with applications:

1.) Mouse mode - Functions 1 - 36 are functional equivalents of Microsofts' MOUSE.COM specification. Interfacing to these functions allows the tablet to emulate a relative pointing device.

2.) Absolute mode - Function 177 (Mode Select) can change screen cursor tracking from relative to absolute. To maintain screen aspect ratio the tablet's active region should be set through function 178.

3.) Digitizer mode - The application can read absolute co-ordinates from function 180 (Read Absolute Co-ordinate).

4.) Direct I/O mode - Functions 182 - 185 allow the application to interface to the serial I/O portion of the driver and talk directly to the tablet.

The driver can also draw and maintain a screen cursor that will follow the motion of the input device, and execute user-defined subroutines whenever a device interrupt occurs.

1.5 Programming Guidelines

The Microsoft Mouse driver, and compatible drivers such as TABLET.COM, include both basic features and advanced features.

1) *Keep it Simple* - If there is anything we have learned in all aspects of programming it is the advantage of simple designs. Most applications only need to use four functions - function 0 to init the driver, function 3 to poll for positional and button information, and functions 1 and 2 to show and hide the cursor. If your application will be used with multiple pointing devices the less you demand from other drivers, the less likely you are to run into problems.

2) *Isolate Driver Calls* - Use one procedure to access the TABLET.COM driver. This provides an easy way to trace any calls to the driver, and will greatly speed testing and debugging.

3) *Avoid Function 4* - Applications that set the location of the cursor are fundamentally incompatible with tablets and other absolute pointing devices.

4) *Avoid Function 12* - Function 12 creates a process which runs at interrupt level whenever the pointing device generates new coordinates or a button is pressed. Testing and debugging interrupt level processes that interact with application programs is very difficult. For example, you must ensure that the interrupt level process is terminated (by a function 12 or 0 call) before exiting from your program. That requirement alone requires you to add code to your application to handle all DOS error terminations such as "disk not ready", "control-break", etc. And then you have to test all that code, and you'll find that some PCs don't work the same as other PCs at this level, so you'll add more code, and so on.

5) *Avoid using TABLET.COM to draw cursors on EGA boards* - The design of the EGA requires extensive software gymnastics to correctly draw cursors from an interrupt process such as a mouse driver. Basically you must use a set of function calls added to the int 10 BIOS interrupt to manipulate the EGA registers. This slows your screen operations considerably, and adds another layer of complexity which is almost impossible to adequately test.

1.5 Programming Guidelines (cont.)

If you want to tackle this challenge, the chapter entitled "The EGA Interface" in this manual provides the necessary documentation on how to use the driver's EGA functions. TABLET.COM has an EGA Interface that is fully compatible with Microsoft's.

In general you should avoid trying to access the EGA card from interrupt processes, including mouse drivers and TABLET.COM. Instead, we recommend that you use mouse function 3 to get the location of the cursor and plot your own cursor.

Chapter 2
Calling TABLET.COM

Calling TABLET.COM

The resident driver TABLET.COM has two entry points. One is through a software interrupt. This is normally used by assembler language programmers and in support libraries for compiled high level languages. The second entry point is from the BASICA interpreter. The driver also provides emulation of the light pen so that BASICA programmers can use the light pen functions in BASICA with TABLET.COM.

2.1 Software Interrupt

The first entry point to TABLET.COM is through software interrupt 33h (51 decimal). The parameter values required by TABLET.COM are passed in the registers AX, BX, CX, and DX. These arguments are referred to in the remainder of this document as M1%, M2%, M3%, and M4% to retain a nomenclature consistent with Microsoft's. Any values returned by TABLET.COM are returned in the same registers. The following mouse and tablet functions require a long pointer to be passed in ES:DX: 9, 12, 20, 22, 23, 24, 178, 179, 182, 183, 185, 187. Functions 186 and 189 return a long pointer in ES:DX, and function 31 returns a long pointer in ES:BX.

Example: Calling TABLET.COM via Software Interrupt from Assembly Language

```
MOV     AX, 0       ;   reset mouse function
INT     33H         ;   call TABLET.COM
MOV     AX, 10      ;   set text cursor function
MOV     BX, 0       ;   use software cursor
MOV     CX,0FFFFH   ;   cursor is reverse video box
MOV     DX, 3300H
INT     33H         ;   call TABLET.COM
```

As with any call to a loadable driver through the system interrupt vectors, you should check to see that a driver is present before calling the driver or a system crash may occur. In DOS 2.1 it is sufficient to check for a non zero vector. In DOS 3.0 and above you must check to see that the vector is not pointing to an IRET instruction (0CF hex). Of course under DOS 3.0 a call to an uninitialized software interrupt will result in a "null" driver call instead of a system crash. You can verify that the driver is operational by checking the returned values from the "initialize" call, Function 0. It is a good idea to reset the driver with function 0 before making any other calls.

2.2 Basic Interpreter

A second entry point to TABLET.COM is provided for users of interpreted BASIC. This entry point is 2 bytes after the main TABLET.COM entry point and is accessed by reading the pointer from the interrupt vector table at 33 hex (51 decimal), adding two, and using the resulting pointer in the BASIC CALL statement. Note that both a segment value and an offset must be read from the interrupt vector table.

Example: Calling TABLET.COM via BASIC*

```
100     DEF SEG = 0
110     MSEG = 256*PEEK(51*4+3) + PEEK(51*4+2)
120     TABLET = 256*PEEK(51*4+1) + PEEK(51*4) + 2
130     DEF SEG = MSEG
140     IF (MSEG = 0 AND TABLET = 2) THEN GOTO 210
150     IF (PEEK(TABLET-2) = &HCF) THEN GOTO 210
160     M1% = 0 :  M2% = 0 : M3% = 0 : M4% = 0
170     CALL TABLET(M1%, M2%, M3%, M4%)
180     REM  THIS INITIALIZES THE DRIVER
190     REM  BY CALLING WITH FUNCTION 0
200     STOP
210     REM NO DRIVER WAS LOADED!
```

This example includes a check to see if the TABLET.COM driver is loaded before the CALL statement in line number 150. Under DOS 3.0 and above note that you must PEEK at the location pointed to by the contents of TABLET - 2 due to the offset of the BASIC entry point from the beginning of TABLET.COM. Check for an IRET instruction (0CF hex). You must also execute the DEF SEG = MSEG instruction before looking for the IRET instruction so that the PEEK operation occurs in the right segment.

As a final note, all of the parameters M1% through M4% must be used in every CALL, even if some of them are not needed by TABLET.COM. The parameters must also be integer variables, not constants, strings, floating point numbers, or other data types.

Both the interrupt and BASIC entry points expect integer values for the parameters and almost no checking is performed on the ranges of the input values. Unpredictable results may occur with out of range parameters.

* *Note: This example was written in IBM's BASICA. Microsoft's GW BASIC will also work. For mouse driver information regarding Microsoft Quick Basic or other DOS-based BASIC packages, refer to the user manual shipped with the particular software.*

pter 3
BLET.COM and the Video Display

TABLET.COM and the Video Display

The mouse portion of the driver keeps track of the video mode by capturing the BIOS interrupt 10H vector. As long as you use this BIOS function to set the video mode, the mouse driver can draw an on-screen cursor and keep track of the cursor position by a special coordinate system. TABLET.COM uses the coordinate system specified by the Microsoft standard.

In 80 x 25 text modes, each character is 8 x 8 driver coordinates. Across the top line, character coordinates are (0,0) (8,0) etc. The character on the bottom left of the screen is at (0,192). In 40 x 25 text modes, each character is 16 x 8 driver coordinates.

In low resolution, 320 x 200, pixel graphics modes, each pixel is 2 coordinates wide and 1 coordinate deep. As with the text modes, the upper left pixel is (0,0).

In high resolution graphics modes such as 640 x 200, 640 x 350, or 640 x 480 pixels, the driver coordinates correspond to the actual pixel coordinates (each pixel is 1-by-1).

3.1 The Hercules Graphics Adapter

The size of the Hercules coordinate grid is 720 across by 348 down (the coordinates are equal in size to the pixels). The Hercules monochrome graphics card never was never officially supported by IBM, and lacks BIOS support. The display setup for Hercules graphics modes is not done via interrupt 10h as are other standard display modes. The driver does support Hercules graphics, but cannot detect Hercules graphics modes with its interrupt 10h stub. Instead, your program should use the following Microsoft "standard" procedure to establish a Hercules mode with the mouse driver:

1. Put the Hercules display adapter into graphics mode by setting the appropriate registers (consult your adapter's manual).

2. Store a value in the BIOS data area variable at 40:49h that indicates which Hercules page you are using. Store a 6 for page 0, or a 5 for page 1.

3. Now reset the driver using mouse interrupt 33h function AX=0. The driver now knows that you are in a Hercules mode.

TABLET.COM knows that you have left Hercules mode when a "legitimate" video mode is established via Interrupt 10H.

apter 4
blet Functions

4.1 Function 176: Hardware Interface Details — BASIC

INPUT:

M1% = 176

OUTPUT:

M1% = 0
M2% = base I/O address of hardware
M3% = IRQ # used by hardware

This function returns parameters used by the tablet interface. In the usual case of an RS-232 interface, M2% is the first I/O address claimed by the UART, and M3% is the IRQ used by the serial port.

4.2 Function 177: Mode Select — BASIC

INPUT:

M1%	=	177
M2%	=	0 for relative mode
	=	1 for absolute mode
M3%	=	0 - use current region
	=	1 - reset current region to default
M4%	=	0 - return buttons as a bit vector
	=	1 - return button number

OUTPUT:

M1% = -1 to indicate the tablet functions are supported

This function selects between absolute and relative coordinate systems. The default condition is relative coordinates, i.e., normal mouse emulation. Once absolute mode is selected, however, it will remain active until this function is used to reset it to relative mode. This is done to allow a "control panel" to set absolute mode operation for programs that are normally used in relative mode.

The default region is the entire tablet active area The tablet active area is directly correlated, or mapped, to the computer screen.

The final argument selects between button information being returned as a bit vector of 16-button bits, and button information being returned as an integer which represents the index number of the first button pressed. The index value 0000 represents no buttons pressed. The default value for returning index values is 1.

The current region is set by function 178 and can be read by function 179. The absolute/relative status and button report mode can be ascertained with function 181.

4.3 Function 178: Set Active Region BASIC

INPUT:
M1% = 178

For Assembler...
ES:DX = long pointer to an array of unsigned integers:
Lower left x
Lower left y
Upper right x
Upper right y

For Basic...
M4% = the address of the array

OUTPUT: none

This function is used to define an active area on the tablet surface that behaves like a smaller tablet. In absolute mode the position of the pointer within the active region is scaled to get the position of the mouse cursor hotspot within the virtual screen. In either absolute or relative mode, when the pointer is outside the active region, the driver treats it as if it were out-of-proximity.

Note that the arguments follow the natural coordinate system of a tablet with the origin in the lower left. This differs from the arguments to mouse function 16, for example, where the origin is in the upper left.

4.4 Function 179: Get Active Region BASIC

INPUT:
M1% = 179

For Assembler...
ES:DX = long pointer to an array of unsigned integers to write into:
Lower left x
Lower left y
Upper right x
Upper right y

For Basic...
M4% is the address of the array

OUTPUT:
The array values are written by the driver.

This function is used by an application to determine the coordinates of the active area on the tablet. Initially these are set to the physical boundaries of the tablet. This function can also be used by "popup" programs that alter the active area and then restore it when they exit.

4.5 Function 180: Read Absolute Coordinate BASIC

INPUT:

M1%	=	180

OUTPUT:

M1%	=	Button Status Word, bit vector or index number.
M2%	=	X Coordinate, 16 bit unsigned integer.
M3%	=	Y Coordinate, 16 bit unsigned integer.
M4%	=	Z Coordinate, 16 bit unsigned integer. Set to -1 if the tablet is reporting out of proximity.

The coordinates returned by this call are relative to the active area, with an origin in the lower left corner of the region defined by the set active area. If the default active area is used, these coordinates are equal to the tablet coordinates. The driver always works with coordinates in the first quadrant to avoid the complexity an applications author faces if signed coordinates and a movable origin are allowed.

Absolute coordinates can also be read via the standard function 3 call in scaled screen coordinates once function 177 has been used to select absolute mode.

4.6 Function 181: Query Tablet Configuration BASIC

INPUT:

M1%:	=	181

OUTPUT:

M1% High byte:	=	0 for relative mode
	=	1 for absolute mode
M1% Low byte:	=	0 for buttons as bit vector
	=	1 for button number
M2%	=	Number of buttons available on the cursor
M3%	=	Tablet units per inch (or millimeter)
M4% High byte	=	0 if current active area is default
	=	1 if the active area has been changed
M4% Low byte	=	0 for resolution in lines per inch
	=	1 for resolution in lines per millimeter

This function allows an applications program to ascertain the operating mode of the driver and the physical dimensions of the coordinates returned through function 180.

4.7 Function 182: Get Last Valid Packet — ADVANCED

INPUT:
M1% = 182

For Assembler...
ES:DX = long pointer to a buffer big enough to hold a complete packet.

For Basic...
M4% is the address of an array that will be used as a buffer.

OUTPUT:
The last valid packet from the tablet will be in the buffer.

This function allows programmers writing special purpose applications to use the serial interface portion of the driver directly. It is particularly useful for programs that alter the coordinate space of the tablet. Note that it will not necessarily function correctly if the tablet's output format is changed in such a way that the synchronization character or flag changes from that used by the driver!

4.8 Function 183: Select I/O Mode — ADVANCED

INPUT:
M1% = 183
M2% = 0 for coordinate capture (normal mode of operation)
= 1 for direct I/O
M3% = Maximum number of bytes to transfer to buffer. (unsigned)

For Assembler...
ES:DX = pointer to a buffer of sufficient size to hold M3% bytes.

For Basic...
M4% is the address of an array that will be used as a buffer.

OUTPUT:
The buffer used to capture incoming tablet data characters.

This function is used to read back arbitrary data from the tablet. Incoming characters are simply placed in the buffer as they are received. Function 184 can be used to monitor the number of characters received. If the buffer is not filled to the maximum specified by the M3% parameter, the application should switch back to coordinate capture mode before reading the buffer. If the buffer is filled, the I/O mode automatically reverts back to coordinate capture mode. Note that the driver does not update its position or button status in direct I/O mode.

4.9 Function 184: Number of Characters Received ADVANCED

INPUT:
M1% = 184

OUTPUT:
M2% = number of characters received.

This function is used with function 183 to capture characters from the tablet. The value of "characters received" is cleared whenever function 183 is called with M2% = 1.

4.10 Function 185: Write to Tablet ADVANCED

INPUT:
M1% = 185
M2% = number of characters to send

For assembler...
ES:DX = long pointer to the string of characters to send.

For Basic...
M4% is the address of the string of characters to send.

OUTPUT:
None

This function is used to send commands directly to the tablet. Before issuing this command it is a good idea to invoke direct I/O mode using function 183 so that any output from the tablet will be captured.

4.11 Function 186: Query Tablet ID — BASIC

INPUT:
M1% = 186

OUTPUT:
M2% = TABLET.COM version ID, (High Byte is integer part, Low Byte is hundredths)
For assembler...
ES:DX = pointer to null-terminated ID string

For Basic...
M4% = address of null-terminated ID string

The version number is in BCD format. The string contains tablet specific identifying information such as the manufacturers designation of the tablet. The string is a read only data object.

4.12 Function 187: Set Event Call Address — ADVANCED

INPUT:
M1% = 187
ES:DX = far pointer to event handler code

OUTPUT:
M1% = -1 to indicate completion and that the function is supported.

After registering with TABLET.COM through this function, the event handler code will be executed whenever the conditions of the call mask established by function 188 are met.

4.13 Function 188: Set Event Masks For Call — ADVANCED

INPUT:

M1%	=	188
M2%	=	button event mask: bit 0 for first button, bit 1 for second button, etc.
M3%	=	motion event mask: bits 0, 1, and 2 are set for X, Y, and Z motion event calls, respectively. Bit 15 is set for change-in-proximity-state events. Bits 3-14 are reserved and must be set to all zeros.

OUTPUT:

None

This function is used to manipulate the event masks that control the extended function tablet call. Bits that are set in these masks allow various types of tablet events to trigger calls to the routine established by function 187. If both masks are zero, then all event calls will be disabled. The button mask bits enable both press and release events for the corresponding button.

4.14 Function 189: Get Event Call Information — ADVANCED

INPUT:

M1%	=	189

OUTPUT:

M2%	=	button event mask
M3%	=	motion event mask
ES:DX	=	current event call handler

This function returns information about the current state of the event call auxiliary process. The values returned are those last established by functions 187 and 188.

4.14.1 Parameters for Event Call

ENTRY:

AX	=	current button state
BX	=	current raw X-coordinate
CX	=	current raw Y-coordinate
DX	=	current proximity/Z-coordinate
ES:SI	=	long pointer to a table of additional information in the driver's memory, with the following structure:

```
BUTTONFLAG  DW ?  ; flags set for button events
MOTIONFLAG  DW ?  ; flags set for motion events
LASTBSTATE  DW ?  ; button state the packet before this
LASTXPOS    DW ?  ; X-position after previous packet
LASTYPOS    DW ?  ; last Y-position
LASTZPOS    DW ?  ; last Z-position
RECT_XMIN   DW ?  ; tablet active area: lower left X
RECT_YMIN   DW ?  ;  "    "    "   lower left Y
RECT_XMAX   DW ?  ;  "    "    "   upper right X
RECT_YMAX   DW ?  ;  "    "    "   upper right Y
```

RETURN:

Carry Flag	=	Processing flag. If clear, the driver will continue processing the packet. If set, the packet processing will stop after returning from the user event call.
AX, BX, CX, DX	=	same as on entry. These registers are used by the driver for further processing if the carry flag is clear!

The table of additional information at ES:SI is also used for further processing, but ES and SI do not need to be returned as a pointer to it.

4.14.2 Writing Event-Handling Routines

Writing event-handling routines for drivers is very tricky and something that should only be attempted by advanced programmers. Because of the the need for fast execution time and careful control of flags and registers, we recommend that event handlers be written in Assembly Language. It is important to bear in mind that the call to the routine is made as the consequence of a hardware interrupt which has suspended some other process. One should not make calls to BIOS or DOS services that may not be reentrant.

The fact that the registers returned by the external routine are used in further processing of the tablet information adds an extra degree of trickiness to designing the routine. The routine should explicitly save AX, BX, CX and DX when called and restore them upon exit. Very advanced routines may change the register values or the values in the table at ES:SI in order to affect later mouse processing.

The coordinates passed to and returned from the external routine in the registers are in raw form, i.e. not relative to the active area. The buttons are encoded in bitvector form. The table at ES:SI is located inside the TABLET.COM driver itself and contains additional useful information that the external routine may need.

The first two words in the table, BUTTONFLAGS and MOTIONFLAGS, are arrays of one-bit event flags that are set for which events the driver detected. The meaning of the flag bits is the same as for the M2% and M3% parameters used as event masks by extended functions 188 and 189. The next 4 words in the table are the buttons and coordinates reported by the last tablet packet. These values were compared to those of the present packet to determine if the routine should be called.

Interrupts are enabled when the external subroutine is called. The driver, however, will not reentrantly call the external subroutine. If the routine is still active when another packet is received, the driver will throw away the new packet.

Chapter 5
Mouse Functions

Mouse Functions

5.1 Function 0: Reset Driver

BASIC

INPUT:
M1% = 0

OUTPUT:
M1% = -1 if the tablet is detected, 0 if it is not detected
M2% = number of mouse buttons

This function starts up the driver and resets internal driver variables.

Driver Default State:

Cursor Level	=	-1
Cursor Shape	=	Up-Pointing Arrow
Hot Spot	=	(0, 0), upper leftmost pixel
Text Cursor	=	Inverted Video Box
Call Mask	=	0 (Disabled)
Light Pen Emulation	=	On
Mickey to Pixel Ratio (Horizontal)	=	8 to 8
Mickey to Pixel Ratio (Vertical)	=	16 to 8
Cursor range	=	Full Screen
Cursor position	=	Center of screen
Motion Counters	=	Cleared
Button Press, Release Information	=	Cleared

5.2 Function 1: Show Cursor

BASIC

INPUT:
M1% = 1

OUTPUT:
None

This function increments the cursor level counter in the driver. When the cursor level equals zero, the driver will draw the screen pointer. Successive calls to Function 1 will not increase the cursor level above zero.

The cursor level variable is a way for a program to handle multiple levels of "hiding" the driver cursor. Since the cursor is drawn by the driver whenever the user moves his mouse it is necessary to "hide" the cursor before altering the screen under the cursor. The cursor level counter keeps "hides" and "shows" nested correctly in complex programs.

5.3 Function 2: Hide Cursor BASIC

INPUT:
M1% = 2

OUTPUT:
None

This decrements the cursor level counter in the driver. If the mouse cursor is currently being displayed it will be removed from the screen. Successive calls to Function 2 will continue to decrease the cursor level below zero. Thus you must have a matching "show" for each "hide".

5.4 Function 3: Get Position and Button Status BASIC

INPUT:
M1% = 3

OUTPUT:
M2% = Button Status
M3% = Horizontal Cursor Position
M4% = Vertical Cursor Position

The Button Status is represented by the 3 lowest bits in M2%.

For mice there are two configurations. The Microsoft compatible mice have two buttons: bit zero represents the left button and bit one represents the right button. For each bit a 0 equals "button up" and a 1 equals "button down". Mouse Systems compatible mice have a third middle button, and its state is represented by bit 2.

The coordinates returned in M3% and M4% will be in driver coordinate units as described in Chapter 3: "TABLET.COM and the Video Display". The driver automatically changes its internal operating mode and coordinate system whenever the display mode is changed through a BIOS INT 10H call.

5.5 Function 4: Set Position BASIC

INPUT:

M1%	=	4
M3%	=	New horizontal coordinate
M4%	=	New vertical coordinate

OUTPUT:

None

This function sets the driver cursor to a new position. The coordinates should be within the legal range of coordinates for the type of display screen in use.

In general this function should be avoided since it is incompatible with the absolute pointing capability of a digitizing tablet.

5.6 Function 5: Get Button Press Information BASIC

INPUT:

M1%	=	5
M2%	=	Button number (left=0, right=1, center=2)

OUTPUT:

M1%	=	Button Status (Same format as function 3)
M2%	=	Number of button presses since last call
M3%	=	Horizontal coordinate at last press
M4%	=	Vertical coordinate at last press

This function provides a count of the number of times a button has been pressed. The count is zeroed after each call to the function. Presses are counted separately from releases.

The cursor coordinate returned in M3% and M4% is the coordinate at the moment of the last button press, and may not be equal to the current cursor coordinate.

5.7 Function 6: Get Button Release Information BASIC

INPUT:

M1%	=	6
M2%	=	Button number (left = 0, right = 1, center = 2)

OUTPUT:

M1%	=	Button Status (Same format as function 3)
M2%	=	Number of button releases since last call
M3%	=	Horizontal coordinate at last release
M4%	=	Vertical coordinate at last release

This function provides a count of the number of times a button has been released. The count is zeroed after each call to the function.

The cursor coordinate returned in M3% and M4% is the coordinate at the moment of the last button release and may not be equal to the current cursor coordinate.

5.8 Function 7: Set Horizontal Coordinate Clipping BASIC

INPUT:

M1%	=	7
M3%	=	Minimum Coordinate
M4%	=	Maximum Coordinate

OUTPUT:
None

This function allows you to set the maximum and minimum values for the horizontal position of the cursor. The cursor will be moved into the region if it is not already there.

If the Maximum Coordinate is less than the Minimum Coordinate they will be exchanged.

5.9 Function 8: Set Vertical Coordinate Clipping — BASIC

INPUT:
M1% = 8
M3% = Minimum Coordinate
M4% = Maximum Coordinate
OUTPUT:
None

This function allows you to set the maximum and minimum values for the vertical position of the cursor. The cursor will be moved into the region if it is not already there.

If the Maximum Coordinate is less than the Minimum Coordinate, they will be exchanged.

5.10 Function 9: Define Graphics Cursor — ADVANCED

INPUT:
M1% = 9
M2% = Cursor Active Spot (Horizontal offset)
M3% = Cursor Active Spot (Vertical offset)
M4% = Pointer to Screen and Cursor Mask Bitmaps

(M4% is a long pointer - use ES:DX for assembler interfaces.)

This function allows the programmer to define new cursor shapes for the graphics display modes. Two bitmaps, each 16 bits square are logically combined directly with the display memory to to create a cursor. The Screen mask bitmap is XORed with the result. For EGA display modes the same operation is repeated with all of the planes in the display.

Since the logical operations are conducted directly on screen memory the size of the cursor in pixels varies with the display mode. For example, in 640 x 200 black and white graphics mode the cursor is 16 x 16 pixels. In the 320 x 200 medium resolution color mode the cursor is 8 x 16 pixels.

The active spot offsets specify the relative position of the cursor bitmap to the screen cursor coordinate. The default offset of (0,0) represents the upper left corner of the cursor image. With the default arrow cursor, this is a point just beyond the arrow's tip. The offsets must be in the range - 16 to +16.

From BASIC* create an integer array BITMAP (15,1) where BITMAP (0,0) to BITMAP (15,0) define the screen mask and BITMAP (0,1) to BITMAP (15,1) define the cursor mask. BITMAP (0,X) is the top element as the bitmap is displayed on the screen.

Use the first element of the array as M4%:

```
M1%    =    9
M2%    =    -1
M3%    =    -1
CALL MOUSE (M1%,M2%,M3%, BITMAP (0,0))
```

** Note: This example was written in IBM's BASICA. Microsoft's GW BASIC will also work. For mouse driver information regarding Microsoft Quick BASIC or other DOS-based BASIC packages, refer to the user manual shipped with the particular software.*

5.11 Function 10: Set Text Cursor BASIC

INPUT:

M1%	=	10
M2%	=	Cursor Type
M3%	=	Screen Mask or Scan Line Start
M4%	=	Cursor Mask or Scan Line Stop

OUTPUT:
None

This function is used to define a cursor in text modes. Cursor Type selects either a software cursor if M2% = 0, or a hardware cursor if M2% = 1.

For the software cursor, the Screen Mask, a 16 bit integer, is logically ANDed with the character and attribute bits stored in display memory and then the result is logically XORed with the cursor mask.

The format of a character in display memory is:

Bit 15	1 = Blink, 0 = Normal
Bit 14, 13, 12	Background Color
Bit 11	1 = Bright, 0 = Normal
Bit 10, 9 , 8	Foreground Color
Bit 7-0	Character Code

With appropriate selection of Screen and Cursor masks you can create almost any type of text cursor.

The hardware cursor is normally defined as a single blinking line. For the Color Graphics Adapter each character has 8 scan lines, numbered from 0 to 7 starting at the top. On the monochrome display each character is 12 scan lines high. Thus for the color display the standard single line cursor would be defined by M3%=7, M4%=7.

5.12 Function 11: Read Motion Counters ADVANCED

INPUT:
M1% = 11
OUTPUT:
M3% = Horizontal Counter
M4% = Vertical Counter

This function provides direct access to relative movement of the input device in mouse motion units. Motion units are proportional to the distance that the mouse moves. The ration of units to distance depends on the sensitivity (see functions 26 and 27).

Positive counts represent horizontal motion to the right and vertical motion down. The counts are signed 16 bit integers, and any overflow is ignored. The horizontal and vertical counts are zeroed after each call to this function.

5.13 Function 12: Define User Subroutine ADVANCED

INPUT:
M1% = 12
M3% = Control Mask
ES:DX = Far Pointer To User Subroutine

OUTPUT:
None

This function defines a set of conditions for which a user defined subroutine will be called by the driver. The user defined subroutine is invoked when a mouse interrupt occurs and the conditions defined by the Control Mask are met. The Conditions are:

Control Mask Bit	Condition
0	Cursor Position Changed
1	Left Button Pressed
2	Left Button Released
3	Right Button Pressed
4	Right Button Released
5	Middle Button Pressed
6	Middle Button Released
7 to 15	Unused

Setting a Control Mask bit to one enables the condition. Clearing all bits to 0 will disable all calls. Be certain to disable the call to your user defined subroutine whenever you exit from your application. Otherwise the next mouse motion, whether intentional or not, will cause a call through the now dangling reference to uninitialized memory. This particular programming error can be difficult to detect since the exit from the application can occur due to a rare DOS fault, such as Disk Not Ready, and your application's subroutine may remain resident in memory for some time until another program allocates and uses the memory.

Calling Sequence

Your subroutine is called with:

AX = Event Control Word - Bits set as in the Control Mask with a 1 bit signifying an active event.
BX = Button Status
CX = Cursor Position (Horizontal)
DX = Cursor Position (Vertical)
DI = Motion Counter (Horizontal)
SI = Motion Counter (Vertical)

WARNING:

This function should only be used by experienced programmers. The user subroutine is called at interrupt level from TABLET.COM with interrupts enabled. This means that mouse interrupts will occur while your subroutine is active. TABLET.COM will not recursively call your subroutine, but you must exercise caution when calling TABLET.COM functions from your subroutine due to the possibility of changing internal data in TABLET.COM. Remember also that DOS is not reentrant.

5.14 Function 13: Light Pen Emulation On — BASIC

INPUT:
M1% = 13

OUTPUT:
None

This function turns on emulation of a light pen for users of BASIC. Mouse motion will be used to simulate a light pen. Pressing a button will store the coordinates of the cursor. Each call to BASIC's PEN function will return a set of coordinates.

When no mouse buttons are depressed the PEN function will return a "pen up" status.

5.15 Function 14: Light Pen Emulation Off — BASIC

INPUT:
M1% = 14

OUTPUT:
None

This function turns off emulation of a light pen.

5.16 Function 15: Set Scaling — BASIC

INPUT:
M1% = 15
M3% = Horizontal Motion Units/8 Pixels
M4% = Vertical Motion Units/8 Pixels

OUTPUT:
None

This function sets the scaling factor between the mouse motion units and screen pixels. The scaling factors passed in M3% and M4% define how many motion units correspond to 8 pixels. The default scaling is 8 horizontal and 16 vertical.

The scaling values can range from -32768 to 32767. Negative values will actually reverse the sense of mouse motion!

The scaling values set by function 15 only affect TABLET.COM when it is operating in relative mode.

5.17 Function 16: Conditional Off — ADVANCED

INPUT:
AX = 15
CX = upper x screen coordinate
DX = upper y screen coordinate
SI = lower x screen coordinate
DI = lower y screen coordinate

OUTPUT:
None

This function defines a screen region where the cursor will not be plotted. If the screen cursor is within the region when the call is made, or if it enters the region it will be hidden. A call to function 1 is needed to restore the cursor.

Function 16 allows your software to update a portion of the screen without "hiding" the cursor before each screen update.

Functions 17 - 18
These functions are not used.

Function 19: Set Speed Threshold **BASIC**

INPUT:
M1% = 19
M4% = Speed Threshold

OUTPUT:
None

This function is used by earlier versions of the mouse driver (those prior to version 7) to control the "ballistic gain" of relative mouse motion. This function has no effect on drivers (such as TABLET.COM) compatible with mouse version 7 or later. It is retained only for compatibility.

Function 20: Exchange User Interrupt Vector **ADVANCED**

INPUT:
M1% = 20
M3% = New User Interrupt Mask
ES:DX = Pointer To New User Subroutine

OUTPUT:
M3% = Old User Subroutine Interrupt Mask
ES:DX = Pointer To Old User Subroutine

This function is intended to be used by pop-up utilities and other programs that need to temporarily gain control of the driver. By using this function a pop-up can save the state of the foreground application's Function 12 subroutine and install its own mouse handler.

5.21 Function 21: Get Size Of State Save Buffer — ADVANCED

INPUT:
M1% = 21

OUTPUT:
M2% = Size of buffer required to save mouse driver state variables in bytes.

This function is intended to be used by pop-up utilities and other programs that need to temporarily gain control of the driver. By using this function and functions 22 and 23, a pop-up can save and restore the state of the driver as set by the foreground application.

5.22 Function 22: Save Driver State Vars. — ADVANCED

INPUT:
M1% = 22
ES:DX = Long Pointer to data buffer.

This function copies the internal state variables to a buffer area so that they can be restored by Function 23.

5.23 Function 23: Restore Driver State Vars. — ADVANCED

INPUT:
M1% = 23
ES:DX = Long Pointer to data buffer.

This function restores the internal state variables from the buffer area.

5.25 Function 25: Read Alternate Subroutine Vector ADVANCED

INPUT:

M1%	=	25
M3%	=	Alternate Event Mask, bits 5,6,7

OUTPUT:

M1%	=	-1 if Alt. Subroutine vector is undefined
M3%	=	Alternate Event Mask
BX:DX	=	Alternate Subroutine Pointer

This function is used to read the alternate subroutine vectors established by Function 24 when a pop-up program wants to save the state of the vectors prior to defining its own. If M1% = -1 then the particular vector requested is undefined.

5.26 Function 26: Set Mouse Sensitivity ADVANCED

INPUT:

M1%	=	26
M2%	=	Horizontal Sensitivity
M3%	=	Vertical Sensitivity
M4%	=	Double Speed Sensitivity
		(all Sensitivity parameters are from 0 to 100)

This function changes the sensitivity setting of the mouse.

5.27 Function 27: Read Mouse Sensitivity ADVANCED

INPUT:
M1% = 27

OUTPUT:
M2% = Horizontal Sensitivity
M3% = Vertical Sensitivity
M4% = Double Speed Sensitivity
(all Sensitivity parameters are from 0 to 100)

Mouse Sensitivity

The sensitivity parameters control the effective Dots Per Inch (DPI) resolution of the mouse emulation when TABLET.COM is in relative mode. The Horizontal and Vertical Sensitivity settings control the ratio between the physical motion of the mouse and the number of motion units reported by the driver (see Function 15). As you increase the sensitivity, the number of motion units for a given displacement of the mouse increases. Because the number of pixels that the cursor moves is derived from motion units, the mouse becomes more responsive as the sensitivity increases.

The Double Speed Sensitivity does not work with version 7 or later mouse drivers; it is retained for compatibility only.

The total responsiveness of the mouse cursor is controlled by a combination of the sensitivity parameters and the mickey-to-pixel ratio (see Function 15). Unlike most other mouse parameters that can be set by functions, the sensitivities are not changed by a mouse reset (function 0).

The default Sensitivity for all three quantities is 50, which corresponds to a one-to-one ratio between motion units and "ticks" detected by the mouse itself. A sensitivity of 70 approximately doubles this ratio and a sensitivity of 30 approximately halves it.

5.28 Function 28

This function is not used.

5.29 Function 29: Set Video Ram Page Number — ADVANCED

INPUT:
M1% = 29
M2% = Video Ram Page Number

Function 29 and Function 30 are used by applications that need to access multiple display pages in the video memory. Function 29 sets the value.

5.30 Function 30: Read Video Ram Page Number — ADVANCED

INPUT:
M1% = 30

OUTPUT:
M2% = Video Ram Page Number

Function 30 reads the value set by function 29.

5.31 Function 31: Disable Driver — ADVANCED

INPUT:
M1% = 31

OUTPUT:
M1% = -1 if error
ES:BX = Mouse vector prior to driver installation

This function attempts to unhook the driver's int 10H stub and the mouse hardware interrupt vector. It then disables the hardware interrupts for the input device and returns the old int 33H pointer so an application can restore the vector if desired. Usually the old int 33H vector points to a null handler in the BIOS, or to 0000:0000.

This function will return an error if it cannot unhook its vectors due to an intervening stub by another resident program. Check the error flag!

5.32 Function 32: Enable Driver — ADVANCED

INPUT:
M1% = 32

OUTPUT:
None

This function re-enables the driver and re-hooks the int 10H and hardware interrupt vectors if necessary.

5.33 Function 33: Software Reset — BASIC

INPUT:
M1% = 33

OUTPUT:
M1% = -1 if driver installed
M2% = 2 or 3 (the button count)

This function performs the software part of a function 0 reset: Variables that contain the mouse state are given their default values. The hardware and interrupt vectors are, however, not affected by this function.

5.34 Function 34: Set Language Byte — BASIC

INPUT:
M1% = 34
M2% = Language number
- 0 = English
- 1 = French
- 2 = Dutch
- 3 = German
- 4 = Swedish
- 5 = Finnish
- 6 = Spanish
- 7 = Portugese
- 8 = Italian

OUTPUT:
None

This function stores a byte value that is interpreted by certain Microsoft utilities to set the language used.

5.35 Function 35: Read Language Byte — BASIC

INPUT:
M1% = 35

OUTPUT:
M2% = Language Byte

This function returns the language byte value stored by Function 35.

5.36 Function 36: Get Mouse and Driver Information — BASIC

INPUT:
M1% = 36

OUTPUT:
M2% = Driver version (in BCD)
M3% = Interface Type (high) and IRQ # (low)

This function returns information about the version of the mouse driver and the type of "mouse" being "driven." The driver version is the level of Microsoft driver that TABLET.COM is compatible with. The version bytes are in Binary-Coded Decimal (BCD) format. The high byte is the integer part and the low byte is the decimal part of the version number.

The interface type returned in M3% high byte is interpreted as follows:
1 = 8255-based bus
2 = serial
3 = import bus
4 = PS/2-style mouse port

5.37 Functions 37 - 42

These functions are not used.

5.38 Functions 43 - 45: Ballistic Gain Functions — ADVANCED

The industry-standard Microsoft mouse driver (versions 7.0 and higher) has a feature called ballistic gain. Ballistic gain is a technique of converting pointing device motion to screen cursor motion in a way that is nonlinear. The most useful implementation is to give the mouse "acceleration," so that moving the mouse quickly causes the motion to be amplified, yet moving the mouse slowly allows fine positioning of the screen cursor.

The ballistic mouse driver comes with four built-in gain profiles: Slow, Moderate, Fast, and Unaccelerated. The driver provides a new function, 45, that allows an application to select between these profiles or determine which is currently active. The driver further allows the gain characteristics and even the names of the four profiles to be modified. To this end, the driver provides functions 43 and 44 to write and read a gain profile data structure.

We depart from the use of M1% - M4% to describe these advanced functions.

5.38 Ballistic Gain Functions (cont.) ADVANCED

The gain profile data structure requires 324 bytes of storage, broken down as follows:

4 x 1 byte lengths for each profile definition
4 x 32 bytes of "movement" domain data for gain function
4 x 32 bytes of "factor" range data for gain function
4 x 16 bytes of ASCII data for the name of the profile

This data structure stores four distinct sets of profile information, indexed by a number from 1 to 4. Each element of the structure is iterated four times, once for each profile.

The length of a profile refers to how much of the movement and factor storage has meaning. The length value must be less than 32.

The movement and factor tables have corresponding elements. They define the domain and range of a scaling function that varies with the mouse speed. The movement values are in raw mouse motion units. The factor values are scaling ratios expressed in sixteenths of a unit, e.g. a factor value of 16 represents a 1:1 scaling. These tables are to be defined in ascending order. A mouse motion equal to or greater than a motion table unit but less than the next motion unit in the table will be scaled by the corresponding factor.

Mouse motion generally consists of both X and Y movements. The appropriate scaling factor is determined by the maximum of the absolute values of X and Y motion.

5.38.1 Function 43: Send Driver New Gain Data

Entry:
AX = 43
BX = profile to be active
ES:SI = pointer to profile data

Return:
AX = 0 if operation successful
= -1 if there was a problem

This function defines the mouse ballistic gain for the driver.

5.38.2 Function 44: Get Gain Data

Entry:
AX = 44

Return:
AX = 0 if operation successful
BX = currently active profile number
ES:SI = pointer to profile data inside driver

This function allows a mouse application to access the ballistic gain data currently being used by the mouse driver.

5.38 Ballistic Gain Functions (cont.) ADVANCED

5.38.3 Function 45: Set Or Read Gain Profile

Entry:

AX	=	45
BX	=	index of ballistic profile to set use (1-4)
	=	-1 if function is being used to read current profile index

Return:

AX	=	0 if the operation was successful
	=	-2 if there was a problem (like index out of range)
BX	=	index of current profile (for entry BX = -1)
ES:SI	=	pointer to 16-character ASCII name of profile

Mouse Function 45 is used to determine which of the four ballistic gain profiles is currently active in the mouse driver. The index desired is passed to the mouse driver in BX. Function 45 may also be used to read the index of the currently active profile by passing a -1 in BX. The index ranges from 1 to 4 and refers to which of the tables passed by Function 43 defines the mouse ballistics.

Chapter 6
The EGA Register Interface

The EGA Register Interface

Tablet.com intercepts the video BIOS interrupt 10h in order to determine which video mode is being currently used. If an Enhanced Graphics Adapter, or EGA, is installed in the computer, a special set of interrupt 10h functions is also provided to read and write to the EGA registers. These functions start at AH = F0h (well above the range of the real BIOS functions) and are called the EGA Register Interface. If no EGA hardware is present, the EGA Register Interface is not installed.

The EGA register interface is accessed by using the video services software interrupt INT 10h (16 decimal). The AH register is used to determine which function is being called. The following table summarizes these functions. A detailed description of each function will be provided shortly.

AH (hexadecimal)	Function
F0	read a single register
F1	write a single register
F2	read a range of registers
F3	write a range of registers
F4	read a register set
F5	write a register set
F6	set all registers to default values
F7	define default register values
FA	query for interface version

6.1 Why an EGA Interface?

The reason for having such a thing lies partly in the nature of the EGA hardware and partly in the way that the driver draws its cursor on the screen. The EGA, aside from having a horrendously convoluted register addressing scheme, also suffers from the major defect that most of its registers are "write only." Because you cannot determine what state an EGA register is in by reading its contents, your program must keep careful track of how the hardware is set up. This is fine if your program is the only process using the EGA hardware, but if you are using a driver (which updates its screen cursor as part of its interrupt handler) there is trouble. The driver can interrupt your program at any point and change the EGA registers when it draws the cursor. It cannot restore the state of the EGA because it cannot find out that state from the registers alone.

This is the problem that the EGA Register Interface attempts to solve. The idea is simple: instead of writing values directly to the registers, one always sets the register values by using the special functions provided. In addition to writing the values to the EGA hardware, these functions also copy the values into corresponding memory locations. If a program faithfully uses these functions to set the EGA registers, than the cursor drawing routine can restore the state of the EGA hardware by looking at the registers' memory images.

If you think that this is a little too far to go just to have the input device draw the cursor for you, we agree! The additional software layer provided by the EGA Interface just adds needless processing time to an already slow graphics interface. The "bug potential" of a program that uses interrupt driven graphics is very high. We recommend that for EGA modes you should keep the mouse cursor hidden, use mouse position function 3 to place a cursor drawn by your own routines, and address the hardware directly.

6.2 EGA Registers

The driver's EGA interface organizes the registers into conceptual groups called "ports," which correspond loosely to the chips used by the original EGA hardware. Each port is assigned a number which is used to reference it when you use the EGA functions. The eight valid port numbers are listed in the table below, along with the name, address, and number of registers in the port:

Port	Register Name	# of regs	I/O Address
0	CRT controller	25	3x4h*
8	Timing Sequencer	5	3C4h
10h	Graphics Controller	9	3CEh
18h	Attribute Controller	20	3C0h
20h	Miscellaneous Output Register	1	3C2h
28h	Feature Control Register	1	3xAh*
30h	Graphics Position 1 Register	1	3CCh
38h	Graphics Position 2 Register	1	3CAh

(* x=D for color modes or x=B for monochrome modes)

Note that the first four ports have multiple registers, while the last four are single register ports. The multi-register ports corresponds to a peculiarity in how the EGA hardware is addressed: a whole series of internal registers can be addressed through just two externally addressed registers. To write to one of the internal registers, you must first supply a value to an external "index" register that selects which internal register you want. Then you write to an external "data" register to set the selected internal register. The driver's EGA Interface handles the details of addressing the hardware, the important thing is that you provide an index along with the port number when you address ports 0-18h.

6.3 Special Cases

Palette Registers. The standard way to set the color palette is by using BIOS interrupt 10h, function Bh. With the driver's EGA Interface, however, it is best to use the provided register write functions to change the palette registers, which are located on the Attribute Controller chip (see below). This ensures that the driver software knows about changes to the palette

Attribute Controller. Both the index and data external registers of this multi-register port (18h) have the same physical I/O address (3C0h). The register at this address flip-flops between data and index each time a value is written to it. A read of the Input Status 1 register (color 3DAh, monochrome 3BAh) sets the index/data toggling to the known state of "index."

The EGA Interface assumes that this toggle is always in the index mode, and always leaves it in index mode after it has been called. If your application needs the Attribute Controller to remain in the data state for any period of time, interrupts should be disabled for that period to prevent a reset to the index state by the mouse cursor drawing. Before setting an Attribute Controller register with the EGA Interface, your program should first read the Input Status 1 register (directly!) to set the index state.

Input Status Registers 0 and 1. Reading these registers should be done directly, not through the EGA Interface. Status register 0 is at address 3C2h and Status register 1 is at 3DAh for color or 3BAh for monochrome.

Sequencer Memory Mode and Graphics Controller Miscellaneous Registers. These registers cannot be read from or written to using the EGA Interface. They must be accessed directly with a special procedure to prevent hardware glitches that can affect the video RAM operation. To change one of these registers, follow this procedure:

1. Disable interrupts,
2. Clear the Synchronous Reset bit (#1) of the Sequencer Reset register.
3. Write to the register. The Sequencer Memory Mode register is address 3C5h, index 4. The Graphics Controller Miscellaneous register is adress 3CFh, index 6.
4. Set the Synchronous Reset bit on the Sequencer Reset register.
5. Enable interrupts.

6.4 EGA Function Descriptions

Function F0h: Read a Single Register

INPUT:

AH	=	F0h
BX	=	index for multi-register chips
DX	=	port number (see list above)

OUTPUT:

BL	=	EGA register data

(All other CPU registers preserved)

This function "reads" data from a single EGA register. Note that what it really does is read what was last copied to the memory image of the register by functions F1h, F3h, F5h, or F6h.

Function F1h: Write a Single Register

INPUT:

AH	=	F1h
DX	=	port number (see list above)

for single register chips:

BL	=	data

for multi-register chips:

BL	=	index of register
BH	=	data for register

OUTPUT:

BH and DX are altered, BL and all other registers are preserved.

This function writes a single register to the EGA hardware and also makes a copy of the data value in a memory location corresponding to the register.

Function F2h: Read a Range of Registers

INPUT:

AH	=	F2h
CH	=	start index of register
CL	=	number of registers to read
DX	=	port number (multi-register)
ES:BX	=	pointer to a table for the register data. Must be at least CL bytes in length.

OUTPUT:

The table is filled with register values.
CX is altered, all other registers are preserved.

This function applies only to multi-register EGA ports. The range is a series of registers on the same port that have consecutive indexes. Function F3h is the write counterpart of function F2h. Both functions F2h and F3h would be suited to, for example, changing the palette registers on the attribute controller.

Function F3h: Write a Range of Registers

INPUT:

AH	=	F3h
CH	=	start index of register
CL	=	number of registers to write
DX	=	port number (multi-register)
ES:BX	=	pointer to a table of data for the registers. Must be at least CL bytes in length.

OUTPUT:

BX, CX, and DX are altered, the other registers and the data table are preserved.

This function writes data to a contiguous range of EGA registers from a table of byte values. The register range must be on the same port number and have consecutive indexes. The data is written to a memory image of the registers, as well as to the registers themselves. Function F2h is the read counterpart of function F3h.

Function F4h: Read a Register Set

INPUT:

AH	=	F4h
CX	=	number of registers (>1)
ES:BX	=	pointer to a table of register entries, each with the following four- byte data structure: *Bytes 1-2:* Port number. *Byte 3:* Index value for multi-register ports, or 0 for single-register ports. *Byte 4:* slot for register data

OUTPUT:

The data slots in the table are filled in.
CX is altered, all other registers are preserved.

Function F4h reads a set of registers from their memory image. Since both the port and the index are specified for each register in the table, the registers need not be contiguous or on the same port. Function F5h is the write counterpart of function F4h.

Function FAh: EGA Interface Version

INPUT:

AH	=	FAh
BX	=	0

OUTPUT:

If an EGA Interface is installed:
ES:BX points to a two-byte EGA Interface version number in the mouse driver's memory. This number is BCD encoded. The first byte is the integer part of the version number, the second byte is the fractional part, in hundredths.

If an EGA interface is not present:
BX remains 0.

This function can be used to verify if the mouse driver's EGA Interface is present.