ToolKit II

MouseText

USERS MANUAL

KYAN SOFTWARE INC.
SAN FRANCISCO, CALIFORNIA

Toolkit II

MouseText

Adds a Macintosh-like interface to Apple // programs.

Use this software with an Apple //c or enhanced //e and Kyan Pascal (Version 2.0)

**Kyan Software Inc.
San Francisco, California**

TABLE OF CONTENTS

CONTENTS PAGE

CONTENTS PAGE

CONTENTS PAGE

Notice

Kyan Software reserves the right to make improvements to the products described in this manual at any time and without notice. Kyan Software cannot guarantee that you will receive notice of such revisions, even if you are a registered owner. You should periodically check with Kyan Software or your authorized Kyan Software dealer.

Although we have thoroughly tested the software and reviewed the documentation, Kyan Software makes no warranty, either express or implied, with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Some states do not allow the exclusion or limitation of implied warranties or liabilities or consequential damages, so the above limitation or exclusion may not apply to you.

Copyright

This users manual and the computer software (programs) described in it are copyrighted by Kyan Software Inc. and Apple Computer Inc., with all rights reserved. Under the copyright laws, neither this manual nor the programs may be copied, in whole or part, without the written consent of Kyan Software Inc. and/or Apple Computer Inc. The only legal copies are those required in the normal use of the software or as backup copies. This exception does not allow copies to be made for others, whether or not sold. Under the law, copying includes translations into another language or format.

ProDOS and the MouseText Library are copyrighted programs of Apple Computer Inc. They are licensed to Kyan Software for distribution and use only with Kyan Software products. Apple software shall not be copied onto another diskette (except for archive purposes) or into memory unless as part of the execution of Kyan Software products. When the Kyan Software product has completed execution, Apple software shall not be used by any other program.

Copy Protection

Kyan Software products are not copy-protected. As a result, you can make backup copies and load the software onto a hard disk or into a RAM memory expansion card. We trust you. Please do not violate our trust by making or distributing illegal copies.

Copyright and Licensing Requirements

Portions of the MouseText Toolkit were developed by Apple Computer Inc. and licensed by Kyan Software Inc. for use in this product. These Apple developed modules include the MouseText Runtime Module and certain technical reference portions of this manual. The balance of the manual and all of the routines necessary to interface the MouseText Runtime Module with Kyan Pascal and assembly language programs were developed by Kyan Software.

Kyan Software encourages you to use the MouseText routines found in this Toolkit in your application programs. You will find that most of the new software being developed for the Apple II family incorporates the Macintosh-like interface which these routines support.

If you write software for commercial distribution and use the tools provided by Kyan, it is essential that you observe the copyright and licensing regulations established by Kyan and Apple Computer.

Kyan Software does not require a special license or charge a royalty for commercial use of our Pascal Runtime Library or the MouseText interface modules contained in this Toolkit. We do, however, require you to acknowledge our copyright and comply with the terms outlined in the license agreement found in your Kyan Pascal Users Manual. If you would like more information about commercial use of Kyan software modules, please contact Kyan Software at 415-626-2080.

If you want to incorporate the MouseText Runtime Module, ProDOS or some other Apple Computer software module in your commercial software, you must obtain a license from Apple Computer Inc. These licenses are not expensive and are relatively easy to obtain. For more information on Apple's licensing requirements and procedures, please contact:

Apple Computer Corporate Licensing
20525 Mariani Avenue, M/S 23-F
Cupertino, CA 95014
408-996-1010

Limited Warranty

Kyan Software warrants the diskette(s) on which the Kyan software is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by your proof of purchase.

Disclaimer of Warranty -- Kyan Software Inc.

Except for the limited warranty described in the preceding paragraph, Kyan Software makes no warranties, either express or implied, with respect to the software, its quality, performance, merchantability or fitness for any particular purpose. Some states do not allow the exclusion or limitation of implied warranties or liabilities for incidental or consequential damages, so the above limitations or exclusions may not apply to you. This warranty provides you with specific legal rights. There may be other rights that you may have which vary from state to state.

Disclaimer of Warranty -- Apple Computer Inc.

Apple Computer Inc. makes no warranties, either express or implied, regarding the enclosed computer software package, its merchantability or its fitness for any particular purpose. The exclusion of implied warranties is not permitted by some states. The above exclusion may not apply to you. This warranty provides you with specific legal rights. There may be other rights that you may have which vary from state to state.

Technical Support

Kyan Software has a technical support staff ready to assist you with any problems you might encounter. If you have a problem, we request that you first consult this users manual. We have worked very hard to identify and include in this manual, the answers to questions and problems most frequently encountered.

If you have a problem which is not covered in the manual, our support staff is ready to help. If the problem is a program which won't compile or run, we can best help if you send us a description of the problem and a listing of your program (better yet, send us a disk with the listing on it). We will do our best to get back to you with an answer as quickly as possible.

If you question can be answered on the phone, then give us a call. Our technical staff is available to assist you on Monday through Friday between the hours of 9 AM and 5 PM, West Coast Time. You may reach them by calling:

Technical Support: (415) 626-2080

Suggestion Box

Kyan Software likes to hear from you. Please write if you have suggestions, comments and, yes, even criticisms of our products. And, we do listen. It is your suggestions and comments that frequently lead to new products and/or product modifications.

We encourage you to write. To make it easier, we have included a form in the back of this manual. This form makes it easier for you to write and easier for us to understand and respond to your comments. Please let us hear from you.

Mailing Address: **Kyan Software Inc.**
1850 Union Street #183
San Francisco, CA 94123

This page is supposed to be blank.

A. Introduction

Overview

The MouseText Toolkit is a family of routines which provide an easy means of adding Macintosh-like features to Pascal and assembly language programs. With the MouseText Toolkit, you can write programs for the Apple II which include windows with text display, menu-bars, pull-down menus, and mouse-controlled events.

The MouseText Toolkit consists of a Runtime Module (i.e., a binary file which executes the graphics and mouse commands); a set of programming utility routines (i.e., routines which allow you to incorporate the MouseText procedures in Pascal programs); and, more than 50 MouseText procedures which can be "included" in and called from a Pascal or assembly language program. The Toolkit also contains sample programs which demonstrate the use of the Toolkit routines.

To use MouseText routines, you first declare the MouseText constant and type definitions and the desired MouseText procedures in your application program. Then, you call the MouseText procedures in your program to create the desired menus, text windows, and mouse controls. When you are finished, copy your program onto the disk along with a copy of the Pascal Runtime Library and the MouseText Runtime Module. When run, your program will display pull-down menus, text windows, and respond to mouse or cursor controls.

Hardware Requirements

The MouseText Toolkit requires an Apple IIc or Apple IIe with enhanced character ROM (Read Only Memory). It is not compatible with an Apple II or II+ computer.

A mouse is recommended but is not required. If you do not have a mouse, please refer to the end of this section which describes how to use your Apple II keyboard in "Apple II Mouse Emulation Mode".

The Manual

This manual explains how to use the MouseText Toolkit routines in your application programs and provides technical reference information for each MouseText command.

It is recommended that you read the first two chapters of the manual to familiar yourself with the basic features and use of the MouseText Toolkit. The balance of the manual can serve as a reference guide to be consulted when you have questions regarding specific Toolkit commands or procedures.

MouseText Routines

MouseText is the term used to describe the special cursor display characters which can be produced by the character generator in the Apple //c and enhanced Apple //e computer. The MouseText Toolkit is a set of routines which use these characters and other software techniques to give a Macintosh-like "look and feel" to Apple // programs.

MouseText routines can be used in both Pascal and assembly language programs. These routines enable you to write software which supports:

- cursor selection and display (with or without a Mouse)
- four major kinds of events or actions:
  - Mouse events,
  - Keyboard events,
  - Update events, and
  - Application events.
- pull-down menu control and display:
  - Menu bar selection and display,
  - Menu selection and display, and
  - Menu item selection and display.
- window control and display:
  - Window selection and display,
  - Window dragging and size-changing, and
  - Writing within text windows.

- scrolling windows through documents

The overall operation of MouseText commands and the functionality achieved with the MouseText Toolkit are described in the following sections. The individual MouseText commands are listed in subsequent sections of this manual.

Background

The following sections describe the "Macintosh" environment.

The Desktop

The screen may be thought of as the top of a desk, with different documents on it. The user can put a new document on the desk, scan it line by line, page through it, move it around, remove it from the desktop or merely set it aside for a moment while he looks at another document.

Windows

A window represents a document on the desktop. A window has a rectangular content area, where its primary information is displayed. Minimum and maximum sizes of this area are specified by the programmer. A window also has several optional components:

The drag bar is used to move the window around on the desktop. The drag bar sits atop the content area and contains the title of the window.

The close box (or go-away box) is used to remove the window from the desktop. The close box is a small box located just inside the left edge of the drag bar.

The grow box is used to change the size of the window and therefore the amount of information from the document which can be viewed at one time. The grow box is a small box located in the lower right corner of the content area.

Scrollbars are used to scan or page through a document horizontally and/or vertically; they define a rectangular field of vision within the document.

Figure A-1. Desktop Display with Windows.

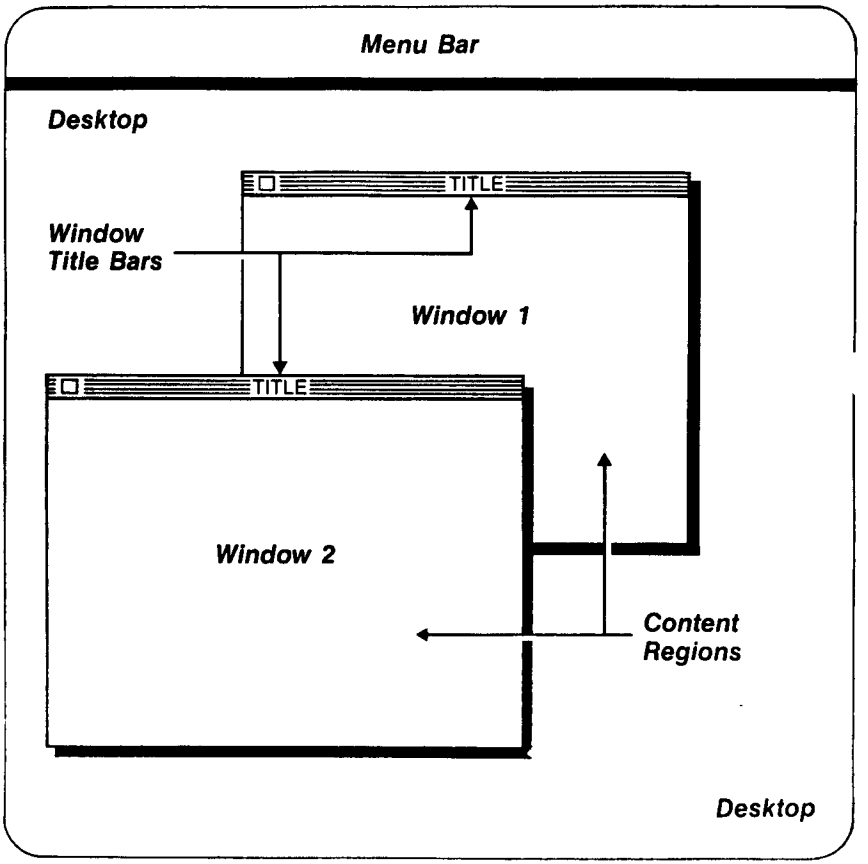
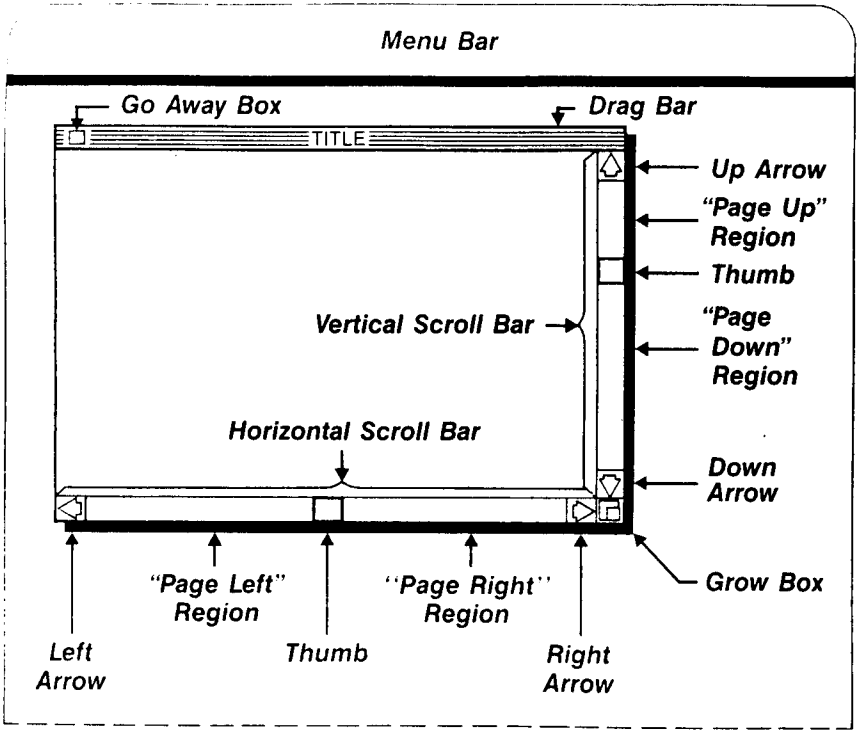


Figure A-2. The Make-Up of a Window.



The MouseText Toolkit is used in applications to do the following when the user clicks the mouse in the control area of the window.

Dragging a Window

The user clicks anywhere in the drag bar (other than in the close box), and drags the mouse. As he does so, an outline of the window follows the mouse to show the tentative new window location. When the user releases the mouse button, the window is released from its starting location and a new, empty window is drawn at the new location. A window can be dragged almost entirely off the desktop, but enough of its drag bar must remain so that the window can be moved back to a usable location.

Closing a Window

The user clicks in the close box. If he also released the mouse button in the close box (even after dragging it outside), the window is released from the screen; otherwise closing is cancelled.

Sizing a Window

The user clicks in the grow box and drags the mouse. As he does so, an outline of the window shrinks or expands to indicate the tentative new window size (the upper left corner of the window remains fixed). If the mouse is dragged beyond the minimum or maximum content size, the cursor continues to move, but the outline does not change size. When the user released the mouse button, a new, empty window is drawn in the appropriate size, and the content area should be redrawn.

Vertical Scrolling

If the user clicks in the thumb and drags the mouse, the thumb will move vertically (within the confines of the scrollbar) along with the mouse until the button is released (even if the mouse is dragged outside the scrollbar). The portion of the document corresponding to the new thumb position should then be drawn.

If the user clicks in the page-up area, the previous 'page' of the document (the area whose last few lines are the same as the first lines of the current page) should be drawn, and the thumb position should be updated. This paging operation should be repeated until the mouse button is released or the first page is reached.

Similarly, if the user clicks in the page-down area, the next page of the document should be drawn, the thumb should be updated, and the operation should be repeated until the button is released or the last page is reached.

If the user clicks in the up-arrow, the document should be scrolled downward by a line (or appropriately small number of lines), and the thumb position should be updated. This scrolling operation should be repeated until the mouse button is released or the first line is reached.

Similarly, if the down-arrow is clicked, the document should be scrolled upward by the appropriate number of lines, the thumb should be

updated, and the operation should be repeated until the button is released or the last line is reached.

Horizontal Scrolling

If the user clicks in the thumb and drags the mouse, the thumb will move horizontally (within the confines of the scrollbar) along with the mouse until the button is released (even is the mouse is dragged outside the scrollbar). The portion of the document corresponding to the new thumb position should then be drawn.

If the user clicks in the page-left area, the previous 'page' of the document (the area whose last few columns are the same as the first columns of the current page) should be drawn, and the thumb position should be updated. This paging operation should be repeated until the mouse button is released or the first page is reached.

Similarly, if the user clicks in the page-right area, the next page of the document should be drawn, the thumb should be updated, and the operation should be repeated until the button is released or the last page is reached.

If the user clicks in the left-arrow, the document should be scrolled to the right by a column (or appropriately small number of columns), and the thumb position should be updated. This scrolling operation should be repeated until the mouse button is released or the first column is reached.

There may be more than one window on the desktop, but there can be only one active window; this window is called the front or top window. A highlighted title, close box, grow box, and fully functional scrollbars appear only for the front window. There is a specific front-to-back ordering of the windows on the desktop, with each window residing in its own plane. This ordering facilitates redrawing windows when the desktop changes.

Each window's location on the desktop plus its size and coordinate system are independent of other windows on the desktop.

Menus

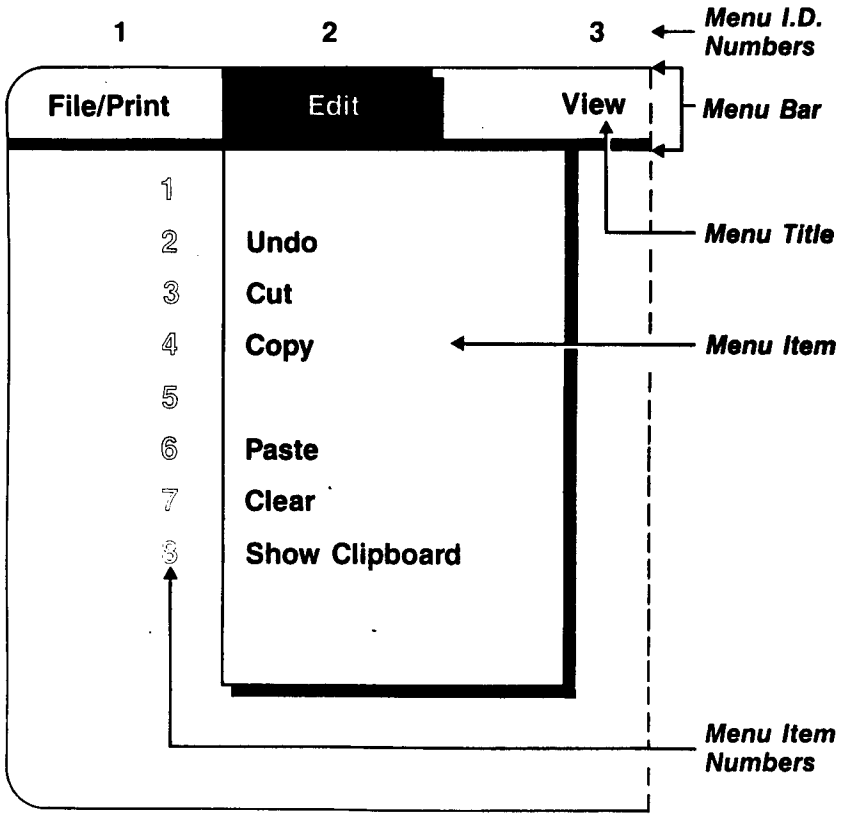
The user makes things happen on the desktop by interacting with pull-down menus and the mouse, and by shortcut keystrokes which execute procedures corresponding to some menu selections. Figure A-3 illustrates the basic components of the menu. The topmost area of the screen is reserved for the menu bar, a line of inverse text listing the menu titles. If the mouse is clicked on a menu title, the title is highlighted and a vertical list of the corresponding menu items appears under it. If the button is released outside the list of items, the list disappears. If the mouse is moved to another title, the list disappears and the list corresponding to the new menu title is displayed. On the other hand, if the mouse is dragged down the list of items, each item will be highlighted while the mouse is over it, indicating that it will be selected if the button is released. If the mouse button is released while a menu item is highlighted, the menu closes and its title stays highlighted until the task associated with the item is complete.

Menu items can be checked off to indicate that certain program features are activated. The checkmark character can be customized for each item.

Menu items, or even entire menus, can be disabled when it is inappropriate to choose them. A disabled item has a different appearance from an enabled item, and does not get highlighted when the mouse is dragged over it. A disabled menu appears with all items disabled.

Groups of menu items can be visually separated by including a special filler item between them. A filler item appears as any character in the filler region or as a solid line.

Figure A-3. Menu Components



The Cursor

The cursor is the image that moves on the display as the user moves the mouse. The programmer specifies the cursor to be an arrowhead, hourglass, checkmark, text cursor, or cell cursor. The Toolkit handles the tracking of the mouse.

Events

A desktop program is driven by events; i.e., something happens and the program responds to it. There are four kinds of events: mouse events, keyboard events, update events, and application events.

> Mouse events are button down, button up, drag (mouse moved while button held down), and apple-key button down (mouse button pressed while an apple key was also pressed).
>
> Keyboard events are keypresses; handling of keyboard events by the Toolkit is optional.
>
> Update events are signals that the contents of a window need to be updated. They are generated when the user re-sizes a window, drags a window, or closes a window. They are not generated when a window is opened or selected.
>
> Application events are specified by the programmer for whatever purposes he deems desirable.

The body of an event-driven program is this loop:

```
Repeat
    Check for occurrence of any event and post it to a queue;
    Get next event to be handled from queue;
    Process event (may include more event-checking);
Until user chooses to quit.
```

The CheckEvents call posts mouse events and keyboard events to an event queue and updates the cursor. The GetEvent call returns the first event from the queue, or, if the queue is empty, returns a null event along with the current mouse position. In passive mode, GetEvent makes a CheckEvents call prior to inspecting the event queue. In interrupt mode, the Toolkit interrupt handler calls

CheckEvents. If the event queue is full, new events will be ignored until there is room for them. The programmer can place any event (except an update event) at the end of the event queue by making a PostEvent call. The queue can be emptied by making a FlushEvents call.

Mouse Emulation

The MouseText Toolkit allows an application to have a Pull-Down Menu/Windowing user interface even if the user does not have a mouse. The Toolkit supports having the user pull down and select items from menus, open windows, close windows, select inactive windows, drag the active window and grow the active window without ever touching a mouse. A brief summary of how this works follows:

Menu Selection. The user pulls down a menu by pressing ESC. The arrow keys then allow the user to move from item to item or menu to menu. The up and down arrow keys move within currently pulled down menu. The left and right arrow keys move from one menu to another. When the user presses left or right arrow, the menu to the left or right is displayed and the mouse pointer moves to the top of the menu. When the user first presses ESC, the mouse pointer appears at the item it was last on. The user accepts a menu selection by pressing either RETURN or any valid key stroke option. The user can cancel the menu selection process by pressing ESC again.

The Windows Menu. To support user's without a mouse, an application must have a Windows menu having the names of all the windows on the desktop. This menu must also have items for Drag, Grow and Hide (with optional key stroke optimizations (APPLE D, G and H).

Opening and Selecting Windows. The user pulls down the windows menu and selects the window s/he wants to open or activate (bring to the top).

Hiding Windows. The user selects "hide" from the windows menu (the top most or active window is hidden).

Dragging Windows. The user selects drag from the windows menu (the top most or active window is available for dragging).

The window is dragged by using arrow keys. Left, right, up and down move the window in the obvious direction. The Apple keys act as amplifiers moving the window further in the obvious direction. The dragging process is terminated when the user presses RETURN, ESC or a valid menu key stroke. RETURN and menu key strokes accept the current position of the window while ESC returns the window to its original position.

Growing Windows. The user selects "drag" from the windows menu (the top most or active window is available for growing). The window is grown by using arrow keys. Left, right, up and down move the window in the obvious direction. The Apple keys act as amplifiers growing the window further in the obvious direction. The growing process is terminated when the user presses RETURN, ESC or a valid menu key stroke. RETURN and menu key strokes accept the current size of the window while ESC returns the window to its original size.

Not supported directly by the Toolkit is any form of text selection. The application is expected to do this on its own. Guidelines for various types of selection are available in the Apple // User Interface Guidelines.

Demonstration Programs

The MouseText Toolkit includes two demonstration programs which illustrate use of the Toolkit and its capabilities and which provide a model for development of your own Toolkit applications. The first program is entitled DEMO2; it is written in assembly language and may be found on Side 1 of the Toolkit disk. The second program is called DEMO1; it is written in Pascal and may be found on Side 2 of the disk. Source code versions of both programs are also included.

You can run DEMO2 by booting Side 1 of the MouseText disk and entering "DEMO2" at the KIX prompt. If you do not have a mouse, follow the instructions outlined above for using the Toolkit in "Mouse Emulation Mode" (i.e., press ESC to pull down a window, use the cursor keys to move through menus, and use the RETURN key to select an item). If you have a mouse, use it to operate the demo program in exactly the same way as you would with a Macintosh.

You will learn more about the MouseText demo programs in the next section.

B. Using The Toolkit

Overview

The MouseText Toolkit provides Pascal or Assembly language programmers with an easy means of developing programs with mouse-controlled menu bars, pull-down menus, and text windows. The Toolkit consists of a Runtime Module, Pascal interface programs, and a number of sample and utility programs.

Runtime Module. The MouseText Runtime Module contains the library of MouseText routines which are called by the MouseText commands. It is a binary file which resides on the disk along with the application program and Kyan Pascal Runtime Library (filename: *MTXKIT.ABS*).

Interface Programs. The Pascal interface procedures consist of small subprograms written in Pascal which call the routines in the MouseText Runtime Module. Each MouseText command has an interface program which must be "included" in your Pascal program whenever you want to use the command. The interface program filename is the same as the command name with a ".I" appended to it.

MouseText Utilities. The utility programs consist of constant and type definition files; an exit routine which allows the programmer to interface application programs with the KIX™ operating environment; and, several other necessary utility files.

When you use the MouseText Toolkit, you must locate the MouseText Runtime Module in the highest available place in memory, normally just below the library. You must also "include" constant definitions, type definitions, and the Pascal-MouseText interface procedures that your program requires (Note: be sure you have a copy of each include file on the program disk.) You then call Toolkit procedures according to your need for menus, text windows and mouse control.

In assembly language programs, you can directly call procedures in the MouseText Runtime Module. This is done in a manner very similar to ProDOS calls. Special interface "include" files are not required.

The Command Reference Section describes how to call each command in assembly language.

A good way to become familiar with the capabilities of the MouseText Toolkit is to study and then run the sample programs provided with the Toolkit. The demonstration program on Side 2 of the disk is written in Pascal and is called DEMO1. The second demonstration program is written in assembly language and is named DEMO2. You are encouraged to use the source code for these sample programs as models for your own application programs.

Toolkit Disk Organization

The MouseText Toolkit is shipped on a "flippy" disk. The volume name of both sides is /MouseText/. The contents of each side are listed in Appendix IV.

Pascal Interface to the MouseText Toolkit

The MouseText Toolkit contains Pascal procedures which are used to interface your Pascal application programs with the MouseText Runtime Module. A Pascal "include" file is provided for each MouseText command. You only need to "include" the procedures that you need for your program; you do not need to waste memory by including all of them.

The Pascal interface procedures are found on side 2 of the MouseText Toolkit disk in a directory called "*PascalTools*". Whenever "including" the Pascal interface procedure in your application program, you must be sure to specify the full pathname of the included file (e.g., #i /MouseText/PascalTools/StartDesktop.I) or be sure that the application program is located in the same directory as the include file (e.g., #i StartDesktop.I).

Strings

The Toolkit expects strings that have the first byte containing the character count. ISO Pascal does not directly support this construct, but one can easily assign the first byte of any string to the total number of characters in the string.

Memory Organization

When using the Toolkit, the runtime memory is organized as follows:

$0	$800	Unavailable.
$800	$1FFF	Heap starts at $800 and grows up. Stack starts at $1FFF and grows down.
$2000	$3FFF	High resolution screen.
$4000	$60FF	Program space.
$6100	$9000	MouseText Runtime Module.
$9000	$BEFF	Kyan Pascal Runtime Library (LIB).

Large Applications

There are several ways of developing large applications using Kyan Pascal and the MouseText Toolkit. With the standard Pascal Library (LIB) and Toolkit loaded, there is 8.25K of space left for the application program. To increase the available program space, one or more of the following options are available.

1. Chain programs together. The demonstration program included with the Toolkit (DEMO1.P) is an example of a chained program. The menus are initialized in DEMO1.P while the main loop is contained in DEMO1A.P. Data is passed between the programs with global variables which are declared in the same order and manner in all chained programs (Please refer to the Kyan Pascal User Manual for more information on Chaining Pascal programs).

2. A Graphics Utility Toolkit is available from Kyan Software which allows you to move your application program into the alternate 64K bank of memory. This technique saves about 12K in the primary bank.

3. The Pascal Runtime Library source files and a code optimizer are available from Kyan Software. By compiling using the Library sources and running the code optimizer, you can compress your object code files by as much as 50 percent.

Data Structures and Constant Definitions

The Pascal interface procedures require specific data structures to function properly with the MouseText Runtime Module. These data structures are already defined in a file named ***MTXKIT.TYP*** found on side 2 of the Toolkit disk.

The disk also contains a file named ***MTXKIT.CON*** where constant values are defined for the maximum:

- size of a menu title string,
- size of a menu item string,
- number of items in a menu, and
- number of menus.

These constants are used in the type definitions to allocate the proper amount of storage for the features used. MTXKIT.CON has default values assigned to each constant. However, you can open this text file and change these default values to suit the needs of your application.

MouseText Runtime Module

The MouseText Runtime Module is 12K in size and is called ***MTXKIT.ABS*** . It can be found on both sides of the Toolkit disk. This Module must be loaded before any Toolkit commands can be executed.

The Toolkit contains a procedure called ***BLOAD.I*** which is used to load the Runtime Module into memory at location $6100. The BLOAD.I file must be "included" in the procedure definition section of your application program (e.g., #I BLOAD.I). The Module is then loaded when you call the procedure in the body of the application program (e.g., BLOAD('MTXKIT.ABS')).

The location of the Runtime Module can be changed using a BASIC program called ***MAKEABS***. This program is used in conjunction with the relocatable file ***MTXKIT.OBJ*** to create a new absolute file. The new absolute file will be a Runtime Module relocated to the address specified. The BLOAD procedure is used in the same way as described above to load this new Runtime Module at the new address.

The Pascal application program must be origined after the high resolution screen. To do this, place the following instructions at the start of your Pascal program (before the Pascal "program" definition).

```
#A
_UsesHires
#
```

Once the Runtime Module is loaded, the Toolkit must be initialized. Then, all MouseText commands are available to the programmer.

Assembly Language Interface

Assembly language programs interface directly with the Toolkit in a manner very similar to ProDOS calls. The same Toolkit Runtime Module (MTXKIT.ABS) is used. This module must be loaded before any Toolkit commands can be executed.

A procedure called ***LOAD.I*** is used to load the Runtime Module into memory. This procedure must first be "included" in the procedure definition section of your application program (e.g., #I LOAD.I). Then, you must call LOAD.I in the body of your program (e.g., LOAD('MTXKIT.ABS')) to load the Runtime Module at location $6100.

Another file, **TOKNIZ.I**, must be "included" in the program before the LOAD.I file. This file sets the proper pathname.

The location of the Runtime Module can be changed using a BASIC program called ***MAKEABS.*** This program is used in conjunction with to the relocatable file ***MTXKIT.OBJ*** to create a new absolute file (i.e., a new MTXKIT.ABS). This new Runtime Module will be relocated to the address specified. The LOAD procedure is used as described above to load this new Runtime Module at the new address.

Once the Runtime Module is loaded, the Toolkit must be initialized. Then, all MouseText commands are available to the programmer.

All calls to the MouseText Toolkit go through a single entry point named Toolkit. In addition to necessary housekeeping functions, the main entry point of the MouseText Toolkit saves the X and Y index

registers and saves the locations in zero-page which it uses for temporary storage.

The programmer can have the assembly language program exit to ProDOS or Kyan's KIX environment by "including" the ***EXIT.I*** routine found on side 1 of the Toolkit disk. This routine is "included" where the application program exits.

The exit routine for the Toolkit also performs housekeeping functions, as well as: restoring the contents of the zero-page locations; restoring the previous contents of the X and Y index registers; and, setting the carry flag to reflect the error status. The exit routine also loads the error status into the accumulator, thereby setting the 6502's N and Z flags.

Syntax of Machine Language Calls

A Machine language call to the MouseText Toolkit looks like the following example:

```
JSR    TOOLKIT    ;main Toolkit entry point
DB     CMDNUM     ;command number of routine called
DW     CMDLIST    ;pointer to parameter list
BNE    ERROR      ;optional error handling
```

After a return from a call to the Toolkit, the value of the program counter is six bytes beyond the location of the calling JSR, and the accumulator contains the error code. The index registers and the stack pointer are unchanged. If the called routine generated an error, the carry bit is on and the zero bit is off; if it did not generate an error, the zero bit is on and the carry bit is off. Table B-1 gives a summary of the return status for Toolkit calls.

Note: Calls to the MouseText Toolkit have the same syntax as calls to the ProDOS Machine Launguage Interface, which is described in the ProDOS Technical Reference Manual.

Table B-1. Processor Status After Return from Toolkit.
A bit value of X means the bit is undefined.

	Processor Status Bits N Z C D V	Accumulator Contents	Program Counter
Successful Call	0 1 0 0 x	0	Calling JSR + 6
Unsuccessful Call	0 0 1 0 x	Error Code	Calling JSR + 6

Final Note: The Assembler included on the Kyan Pascal disk is not intended to provide full support for Assembly Language programming using this Toolkit. For a much more powerful Assembler, we suggest that you try Kyan's **Macro Assembler.**

Programming with the Toolkit

The MouseText Toolkit is designed to run in either the KIX or ProDOS environment. Calls are made via a machine language interface.

One of the easiest ways to write an application using the Toolkit is to start with an existing program and modify it. This Toolkit provides an extensive example of both a Pascal and assembly language application. These sample programs do almost the same things, exercising most of the features of the Toolkit. The examples are also heavily commented to help you understand how they work.

Programs written to use the MouseText Toolkit all have the same flavor. They start with an initialization section and consist of a Main Loop which reads events and handles them according to the kind of event they are. An outline follows describing what a program needs to do.

Initialization

```
Initialize program variables
StartDeskTop        (start up the Toolkit)
InitMenu            (specify special character locations)
Set up menus
ShowCursor          (display the mouse cursor)
```

Main Loop

The main loop is as follows:

REPEAT
  Get an event
  Process the event
UNTIL user quits

Process the Event

If the event is a button-down event:
Find the portion of the screen the mouse is in and initiate the appropriate operation.

If the mouse is in an uncovered area of the desktop:
Do nothing.

If the mouse is in the menu bar:
Call the Toolkit menu selection routine, and execute the menu handling.

If the mouse is in the drag bar of any window:

If the window is not the top most window, select it but do not redraw it. Call the Toolkit DragWindow routine. If the window moved, clear any update events (see below) If the window did not move, and it was selected in step one, redraw it.

If the mouse is in the grow box of the front window:
Call the Toolkit GrowWindow routine, and if the size changed, adjust the scrollbars, and refresh the contents of the window.

If the mouse is in the close box of the front window:
Call the Toolkit CloseWindow routine.

If the mouse is in the content area of the front window:
Find which part of the content area it is in.

If the mouse is in a scrollbar:
Determine which area of the scroll bar it is in and scroll appropriately.

If the mouse is in the thumb:
Call the Toolkit TrackThumb routine, and if the thumb moved, refresh the window to reflect the new thumb position and call UpdateThumb.

If the mouse is in the page-left, page-up, page-right, or page-down area:
Scroll a full window in the appropriate direction and call UpdateThumb.

If the mouse is in the left-, up-, right-, or down-arrow:
Scroll one line in the appropriate direction and call UpdateThumb.

If the mouse is not in a scrollbar:
Execute the appropriate procedure for the window (e.g., selection).

If the mouse is in the content area of an inactive window:
Make it the front window.

If the event is a keypress:
If the key is a shortcut keystroke, find which menu item it corresponds to and execute the menu handling. Otherwise, handle the key according to the context of the application.

If the event is an update event:
Refresh the content area of the specified window.

If the event is an application event:
Handle according to the context of the application.

Menu Handling:

Each menu has a user-assigned menu ID and its menu items are numbered sequentially, beginning at 1. When a menu is selected, the program performs a specific task according to this information, and then de-highlights the menu by calling the Toolkit HiliteMenu routine.

PseudoCode Program Listing

The outline shown above, describing the programming sequence for using the mouse and the MouseText Toolkit, is expanded with the following pseudocode listing of an application program. The pseudocode program is an example of the way the Toolkit is intended to be used. The program illustrates the following functions:

- Start the desktop
- Set up menus
- Set up the cursor
- Track the mouse
- Display a pull-down menu
- Open a window
- Select a window
- Drag a window
- Grow a window
- Scroll the contents of a window
- Close a window

The Pascal demonstration programs DEMO1.P and DEMO1A.P follow the program flow shown below but are not identical. To find out exactly what the sample programs look like, you should list them from the disk.

The user stops this program by selecting the "Quit" item in the menu.

Here is the pseudocode listing of the program

```
call StartDeskTop                    ; start up the Toolkit
call InitMenu                        ; allocate screen save space
call SetMenu                         ; set up our menus
call ShowCursor                      ; turn on cursor
call InitWindowMgr                   ; allocate screen save space for
                                       window
quitflag := false                    ; used to terminate program

while not quit flag do               ; main loop
  call GetEvent                      ; get the next event in event queue
  case eventtype of                  ; base action on type of event returned
     button_up, no_event, drag_event, open_apple_drag_event :
      do nothing                     ; we are ignoring these
    keypress: call HandleKeys        ; handle keyboard input from user
   button_down: call HandleButton ;  handle button down on mouse
  end case
end while                            ; end of main loop
do any clean up
end program                          ; end of program

HandleKeys:                          ; character input is enter here
   if open_apple_key down do         ; check for commands
    call MenuKey                     ; translate into menu command
    call MenuCase                    ; and execute it
end if
return

HandleButton
   call FindWindow                   ; where did button go down ?
   case event_location of            ; base action on where it occurs
     in_desktop: do nothing
     in_menu: call HandleMenu        ; menu bar, menu operation
     in_content: call DoContent      ; content region, find out more
     in_drag_bar: call DragIt        ; drag bar, drag the window
     in_grow: call DoGrow            ; growth region, grow the window
     in_close: call CloseIt          ; close the top window
end case
return
```

```
HandleMenu
   call MenuSelect                ; have Toolkit perform selection
   call MenuCase                  ; execute selection
   return

MenuCase                          ; execute the menu selection
    if menu_id + 0
     then do nothing              ; nothing selected
     else do
       case menu_id & menu_item
         do corresponding operation
       end case
       call HiliMenu(0)           ; task is done, turn off highlight
return

DoContent                         ; button down inside a window
   call FrontWindow               ; find front window id
if button_down does not occur in front window
  then call SelectWindow          ; bring that window to front
  else do
   call ScreenToWindow            ; use local coordinate
   call FindControl               ; find if it occur in control
   case point_is_in
     in_content: depend on application, nothing here
     in_vertical_scroll_bar, in_horz_scroll_bar :
       call ScrollBar             ; perform scrolling
     in_dead_zone: do nothing
   end case
return

ScrollBar
  case where_in_scroll_bar
    arrow, page:
      scroll 1 or n lines
      call UpDateThumb        ; update thumb position
    thumb:
      call TrackThumb         ; let Toolkit track thumb movement
       if thumb_moved then scroll accordingly
end case
return
```

```
DragIt
  call SelectWindow        ; bring window to front if it is in back
  call DragWindow          ; let Toolkit follow the drag
  return

DoGrow
  call GrowWindow          ; let Toolkit follow the growth
  if size_changed do       ; if size of window changed extra work
    call SetCtlMax         ; thumb position,etc may be changed
    call ActivateCtl       ; scroll bar may become active/inactive
    call WinBlock          ; window is blank afterwards, update it
  return
```

Apple II Keyboard Mouse Emulation

Although the menu and window capabilities of the MouseText Toolkit are normally used with the AppleMouse II, it is possible to run a program using the Toolkit on a computer that doesn't have a mouse. It is also possible to use the keyboard to control the menus and windows, even on a computer that has a mouse. When in mouse emulation mode, the Toolkit still responds to movement of the mouse and mouse button operation.

The first method of mouse emulation is called Keyboard Mouse Mode. It enables the application to support menu selection and window manipulation with either a mouse or keyboard commands.

The second method of mouse emulation is called Safety-Net Mode. It is provided specifically for use with a computer that does not have a mouse.

Keyboard Mouse Mode

The Keyboard Mouse Mode of mouse emulation enables applications to substitute keyboard commands for operations that normally require a mouse. The operations which can be performed in this mode are:

- Selecting from a menu,
- Dragging a window, and
- Growing a window.

To perform one of these operations in the Keyboard Mouse Mode, the application program must first call the KeyboardMouse command. This command has no parameters; its purpose is to instruct the Toolkit to perform the next command(s) in Keyboard Mouse Mode. Once in this mode, the Toolkit commands are called in the normal manner (i.e., MenuSelect, DragWindow, or GrowWindow).

There is an alternative way for the application to get into Keyboard Mouse Mode, and that is by calling the MenuKey command with ESC as the keystroke. This has the same effect as calling KeyboardMouse followed by MenuSelect (i.e., it initiates a menu select operation in Keyboard Mouse Mode).

The choice of keypress commands for mouseless operations is specified in the application program by the programmer. While you can choose any keypress sequence you desire, the recommended key sequences are:

Table B-2. Keyboard Mouse Commands

Command	Operation
ESC	display menu
OPEN-APPLE-D or SOLID-APPLE-D	drag a window
OPEN-APPLE-G or SOLID-APPLE-G	grow a window

When the Toolkit is in the Keyboard Mouse Mode, it is performing one of these three operations. It remains in Keyboard Mouse Mode until the operation is completed. Unlike the Safety-Net Mode, the user doesn't have to hold a key down.

When the user initiates the Keyboard Mouse Mode, the Toolkit makes the cursor visible, even if it was previously hidden or obscured. When the keyboard operation is completed, the Toolkit returns the cursor to its previous state of visibility.

When a menu is selected, the Toolkit records the position of the cursor (i.e., the item that is highlighted) and returns to that position (and item) when the user selects the menu again.

In Keyboard Mouse Mode, the cursor keys move the cursor around on the display. If the user is doing a drag or grow, the OPEN-APPLE key acts as an accelerator for the cursor keys. With the OPEN-APPLE key down, pressing left or right arrow keys moves the cursor sideways by 10 spaces at a time. Likewise, the up and down arrow keys move the cursor up and down 5 rows at a time.

The user can terminate a Keyboard Mouse Mode operation in four different ways:

1. Pressing the ESC key.

   The Toolkit cancels the operation and returns the cursor to its former position.

2. Pressing the RETURN key.

   The Toolkit completes the operation and returns the cursor to its former position.

3. Pressing a valid command key

   The Toolkit terminates the operation and then posts an event for the command key. If the operation was a menu selection, the Toolkit cancels the operation. If it was a drag or grow window, the Toolkit completes the operation. In all cases, the Toolkit returns the cursor to its original position.

4. Pressing and releasing the mouse button.

   The button up event signals completion of the operation. It initiates execution of the selected command, just as if the mouse had been used throughout.

Safety-Net Mode

The Safety-Net Mode is intended specifically for computers which do not have a mouse. In this mode the Toolkit uses inputs from the keyboard in place of the usual mouse inputs (i.e., inputs received from mouse movements of the cursor around on the desktop and selection of menus).

When the Toolkit is in Safety-Net Mode, the application program works normally; all command calls are the same. The program need not take into account the fact that there is no mouse.

The user puts the Toolkit into Safety-Net Mode by pressing and holding down the OPEN-APPLE key and then pressing and releasing the SOLID-APPLE key. The Toolkit generates a click to acknowledge that it is in Safety-Net Mode. The Toolkit remains in Safety-Net Mode as long as the user continues to hold down the OPEN-APPLE key.

In Safety-Net Mode, the cursor keys take the place of the mouse in moving the cursor. Each time you press a cursor key, the cursor moves one space in the direction indicated on the key. The cursor keys do not have wrap-around; when the cursor has moved to the edge of a screen, pressing the same cursor key will have no effect.

In Safety-Net Mode, the SOLID-APPLE key takes the place of the mouse button. Pressing the SOLID-APPLE key is like pressing the mouse button.

Note: In Safety-Net Mode, the Toolkit reads the cursor keys and the SOLID-APPLE key even if the application program has specified that the keyboard is to be ignored.

C. Startup and Cursor Commands

Startup Command Overview

Startup commands are called in the application program to set up the operating environment for the MouseText Toolkit. For example, your application program will call the StartDeskTop command to activate the mouse and set the Operating Mode for the Toolkit. Later, it will call the StopDeskTop command to deactivate the mouse and the Toolkit.

Pascal programs can also call the PascIntAdr command to get the address of the Toolkit's interrupt handler so that the Pascal interface can install a custom interrupt handler.

Startup Command List

No.	Name	Description
0	StartDeskTop	Activate mouse and Toolkit routines.
1	StopDeskTop	Inactivate mouse and Toolkit routines.
17	PascIntAdr	Get interrupt handler address for Pascal.
47	SetUserHook	Set address of interrupt handler.
19	Version	Return Toolkit revision numbers.
48	Keyboardmouse	Condition Toolkit to perform next operation in emulation mode.

Startup Programming Notes

Follow this sequence of steps to start the mouse:

(1) (For Pascal only) Call PascIntAdr to get the address of the Toolkit's interrupt handler.

(2) (For Pascal only) Pass the interrupt address to the mouse firmware by calling SetMouse as described in Appendix II, "The Mouse Firmware Interface." Mouse Mode should be set to passive.

(3) Call Startdesktop with the UseInterrupts parameter set the way you want it for your program.

(4) (Optional) Call SetUserHook to pass the addresses of your program's interrupt handlers, if any, to the Toolkit.

The Toolkit saves the interrupt state of the machine when your program calls the StartDeskTop command. When the program calls the StopDeskTop command, the Toolkit sets the state of the machine to the state saved by StartDeskTop.

When you use the Toolkit in Interrupt Mode, The Toolkit provides the interrupt handler. In addition, the Toolkit allows the application program to have interrupt handler subroutines that are called by the Toolkit. The program passes each subroutine's address to the Toolkit as a parameter by calling the SetUserHook command. This feature makes it possible for the application program to perform tasks at interrupt time.

A user hook routine that is called at interrupt time cannot call most Toolkit commands. Doing so could put the Toolkit into an unknown state. If a program needs to generate calls to the Toolkit because of an interrupt, the interrupt routine should set a flag that the program checks during its main polling loop.

Cursor Command Overview

The cursor is a symbolic character that moves on the display screen as the user moves the mouse. Cursor commands allow the programmer to select which MouseText character will be displayed as the cursor and to turn the cursor on or off.

The MouseText Toolkit can use either a mouse-controlled cursor or a keyboard-controlled cursor. Apple II Keyboard Mouse Emulation is described in Section A.

Cursor Command List

No.	Name	Description
2	SetCursor	Sets the character used for displaying the cursor
3	ShowCursor	Makes the cursor visible
4	HideCursor	Makes the cursor invisible
44	ObscureCursor	Makes the cursor invisible until the mouse moves

This page left blank for your notes.

D. Event-Handling Commands

Overview

Events are the means by which a user communicates with the MouseText Toolkit. Events can be mouse "clicks", keyboard keypresses, or other actions. Whatever form an event takes, it is a signal to the Toolkit to initiate a sequence of actions (e.g., display a menu).

Types of Events

The MouseText Toolkit deals with four major kinds of events:

Mouse Events
- o Mouse button pressed down
- o Mouse button released
- o Mouse moved with the button held down (dragging)

Keyboard Events
- o Keypresses

Update Events
- o Special case events used in applications with windows that can't be refreshed automatically.

Application Events
- o Optional events which can occur within an application program.

Precedence of Events

If the mouse button is down, the Toolkit ignores keypresses. Thus, mouse events have precedence over keyboard events.

Event Queue and Processing

The Toolkit's Event-Handling commands maintain an event queue for mouse and keyboard events. The CheckEvents command posts events in the queue and updates the mouse position. The GetEvents command puts the most recent event into the queue. If events occur simultaneously, each event is stored in a queue until it is called for processing.

Event-Handling Command List

No.	Name	Description
5	CheckEvents	Reads the mouse, moves the cursor to the new position, and posts event, if any.
6	GetEvent	Gets next event; if none, gets mouse position.
46	PostEvent	Posts an event in the event queue.
7	FlushEvents	Empties the event queue.
8	SetKeyEvent	Specifies whether Toolkit handles keyboard event.
21	PeekEvent	Returns event data without removing it from the queue.

Event Programming Notes

1. The CheckEvents command posts mouse events and keypress events in the queue and updates the mouse position. To detect mouse events, the program calls the GetEvent command. CheckEvents is not normally used because it is called automatically by GetEvent in passive mode and by the Toolkit in interrupt mode.

2. With the Toolkit running in the Passive Mode (Refer to Appendix I for a description of "Passive" and "Interrupt" Modes), the GetEvent command automatically issues an internal call to the CheckEvent command. If the event queue is empty, the GetEvent command simply returns the most recent mouse position.

3. In the Interrupt Mode, the Toolkit's interrupt handler calls the CheckEvent command 60 times per second, synchronized with the display vertical blanking (VBL).

4. The application program must call the CheckEvent command or GetEvent command often enough to obtain smooth cursor motion.

5. The CheckEvent command is the only command that reads the mouse -- the cursor will never move if the CheckEvent command is never called. The CheckEvent command can be called directly, indirectly (through the GetEvent command), or by the Toolkit itself (in the Interrupt Mode).

6. The application program can put its own events into the event queue by calling the PostEvent command.

7. If the event queue fills up, the Toolkit ignores new events until there is room in queue. To empty the queue, the program can call the FlushEvents command.

8. Frequent calls to the CheckEvents command provides the program with a type-ahead feature. By posting keyboard events in the event queue, they can be stored until the program is able to process them.

This page left blank for your notes.

E. Menu Commands

Overview

Menus are the technique by which a user can view and select a command option. Rather than having to remember a keyboard command structure, the user simply moves the cursor to a menu title, "pulls" or "pops" down the menu, and selects a command from the alternatives listed.

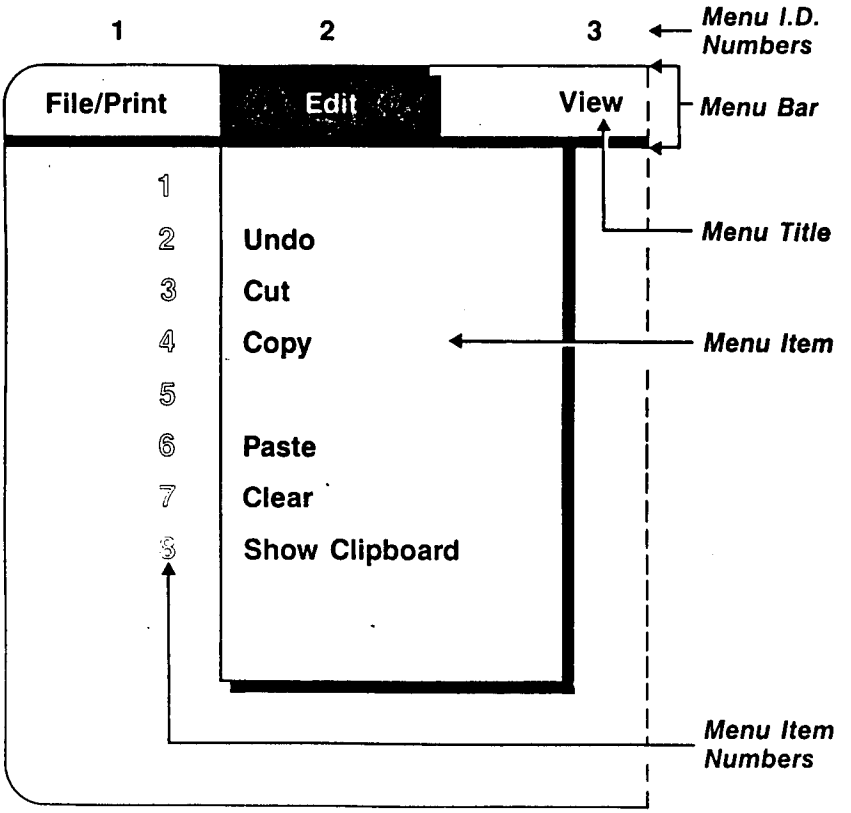
The user doesn't literally pull a menu down. Instead, the menu "pops" down automatically when the application program determines that the user has moved the cursor onto a menu title and pressed the mouse button. As the user moves the cursor down the menu (with the mouse button still depressed), the Toolkit highlights each menu item in inverse video as the cursor passes over it.

When the user releases the mouse button on top of a particular menu item, that item is selected. The menu then disappears. To show the user that something is happening, the Toolkit leaves the menu title in the menu bar highlighted. The title highlighting is turned off as soon as the application program finishes performing the selected operation.

Figure E-1 displays the components of a menu. The visible components are a menu bar (which appears at the top of the display and shows the menu titles) and menu items (which appear, one to a line, when a menu pops down). The invisible components are the menu ID numbers and the menu item numbers.

Menu ID numbers can be in any order as long as they are between 1 and 255. Menu item numbers are always sequential starting with 1. Both numbers are returned to the application program when a menu item is selected.

Figure E-1. Menu Components.



The Toolkit Menu Commands provide menu display and selection functions. Once the menu data structure has been set-up with the SetMenu command, the MenuSelect command will allow the following actions:

- o Display a menu
- o Track the mouse and move the cursor
- o Highlight menu items when the cursor moves to them
- o Return with the menu ID and item numbers selected
- o Leave the menu title highlighted

Other menu commands inhibit menus or menu items and display a checkmark beside specified menu items.

Menu Command List

No.	Name	Description
9	InitMenu	Allocates memory for temporary screen save.
10	SetMenu	Initializes a menu bar data structure and displays the menu bar.
11	MenuSelec	Interacts with mouse to display menu and return selection, if any.
12	MenuKey	Selects a menu item to match a keypress.
13	HiLiteMenu	Turns highlighting of menu title on or off.
14	DisableMenu	Inhibits highlighting and selection over a whole menu.
15	DisableItem	Inhibits highlighting and selection of a menu item.
16	CheckItem	Turns checkmark next to item on or off.
20	SetMark	Sets the character to use as checkmark.

Programming Notes

Keys in Menus

The MenuKey command allows programmers to use keypresses to select menu items. You use a combination keypress consisting of a letter key plus one of the Apple keys. Menu items which can be selected in this way are indicated by the OPEN-APPLE or SOLID-APPLE icon and the specified letter or other key displayed to the right of the menu item. If an item can be selected using either type of apple icon, the OPEN-APPLE icon appears with the letter in the menu.

You can also specify a control character as the keypress which selects a menu item. This is done by setting either Character 1 or Character 2 in the Menu Item Block to any value from 1 to 31, corresponding to a control character. (Menu Item Blocks are defined in the Technical Reference Section under the SETMENU command.) You do not need to set the modifier bits in the Item Option Byte.

When you specify a control key to select an item, the Toolkit displays a diamond icon and the key to the right of the menu item. Only the value in Character 1 will be used, even if you designated Character 2 a control character.

Keypresses with the CONTROL key are easier to touch-type than those with the Apple keys. You should continue to use the Apple-key combinations though, for most items, and reserve the use of control keys for high-speed or repetitive functions where the touch-type command is important.

Users expect control keys to be used for the same functions for different products. Table E-1 shows the menu functions which Apple has defined for most of the control keys.

When you press a key other than one of those specified in the menu, the Toolkit generates a beep.

Table E-2. Control keys for Menu Items

Control Key	Function
CTRL-B	Boldface
CTRL-C	Copy
CTRL-D	Delete
CTRL-E	Editing type, insert or overstrike cursor
CTRL-F	Forward delete
CTRL-H	Left arrow
CTRL-I	Tab
CTRL-J	Down arrow
CTRL-K	Up arrow
CTRL-L	Begin or end underline
CTRL-M	Return
CTRL-P	Print
CTRL-U	Right arrow
CTRL-V	Paste
CTRL-X	Cut
CTRL-Z	Zoom
CTRL-[	Escape

Other Notes

1. When the user moves the cursor onto a title in the menu bar and presses the button on the Mouse, the application program calls the MenuSelect command. This command displays the selected menu and tracks the mouse as long as the mouse button stays down.

2. An application program can disable individual menu items or an entire menu. Disabled items or menus are not highlighted when the cursor moves over them, and they cannot be selected by the user.

3. The SetMenu command is used in the application program to supply the Toolkit with the data structures needed to display menus. The program can call SetMenu during the course of operation to change the contents of menus (menu data structures are described in the Technical Reference Section).

4. The FindWindow command is used to detect the pressing of the mouse button in row 0 (the menu bar). When this event is detected, the application program calls the MenuSelect command.

5. The MenuSelect command takes care of the entire menu selection process. It displays the menus and tracks the mouse cursor position for as long as the user holds down the mouse button. If the user selects a menu item, the MenuSelect command highlights the menu's title in the menu bar and returns the menu item number and menu ID number to the application program. If the user doesn't select a menu item, the MenuSelect command returns a menu ID value of 0.

6. Highlighting of the selected menu title while the selected operation is being performed provides useful feedback to the user. When the operation is complete, the menu title is "un-highlighted" by calling the HiLiteMenu command with the menu ID set to 0.

7. For menu items that are used often, the application program can provide the user with fast item selection. This is done by defining keypress equivalents for the items in the menu and giving the user an option to press keys instead of moving the mouse. When the GetEvent command returns a keypress, the application program calls the MenuKey command. MenuKey determines the menu ID and item number by searching the menu data structures for a matching key. When it finds a match, it highlights the selected menu title the same way MenuSelect does. After the operation has been performed, the program uses the HiLiteMenu command to turn off the highlighting.

8. The application programmer must ensure that menu titles do not extend past the right edge of the screen. The programmer must make sure that a menu's width is always less than the screen width minus two (38 or 78), and that a menu's length is always less than screen length minus two (22). Otherwise, the menu routines can write into main memory when they should be writing to the display, thereby clobbering screen holes or program memory.

F. Window Commands

Overview

The Toolkit Window Management commands provide the functions needed to set up and display windows. When the window information structure has been set up with the OpenWindow command, you can use these commands to:

- o select a window
- o bring the window to the front of the display
- o put text into the window
- o drag the window
- o change the window size
- o close the window.

Window Command List

No.	Name	Description
22	InitWindowMgr	Initializes the open window list and buffer area.
23	OpenWindow	Passes the Toolkit a pointer to a Window Information Data Structure.
24	CloseWindow	Deletes a window.
25	CloseAll	Deletes all windows.
45	GetWinPtr	Gets the Pointer to the Window. Information Data Structure (not applicable to Pascal).
26	FindWindow	Finds the window region that contains a given name.
27	FrontWindow	Returns the ID number of the front window.
28	SelectWindow	Makes a window the front (active) window.
29	TrackGoAway	Returns whether the mouse button was released in a Go-Away Box.

Window Command List (cont.)

No.	Name	Description
30	DragWindow	Displays window outline during drag, then redisplays window.
31	GrowWindow	Displays window outline during grow, then redisplays windows.
32	WindowToScreen	Converts window coordinates to screen coordinates.
33	ScreenToWindow	Converts screen coordinates to window coordinates.
34	WinChar	Writes a character in a window.
35	WinString	Writes a string in a window.
38	WinText	Writes text in a window.
36	WinBlock	Writes a block of text in a window.
37	WinOp	Clears all or part of a window.

Programming Notes

Components of the Window

Window commands in the MouseText Toolkit make it possible for programs to use the mouse to control multiple windows on the desktop. Figure F-1 shows the various aspects of MouseText windows.

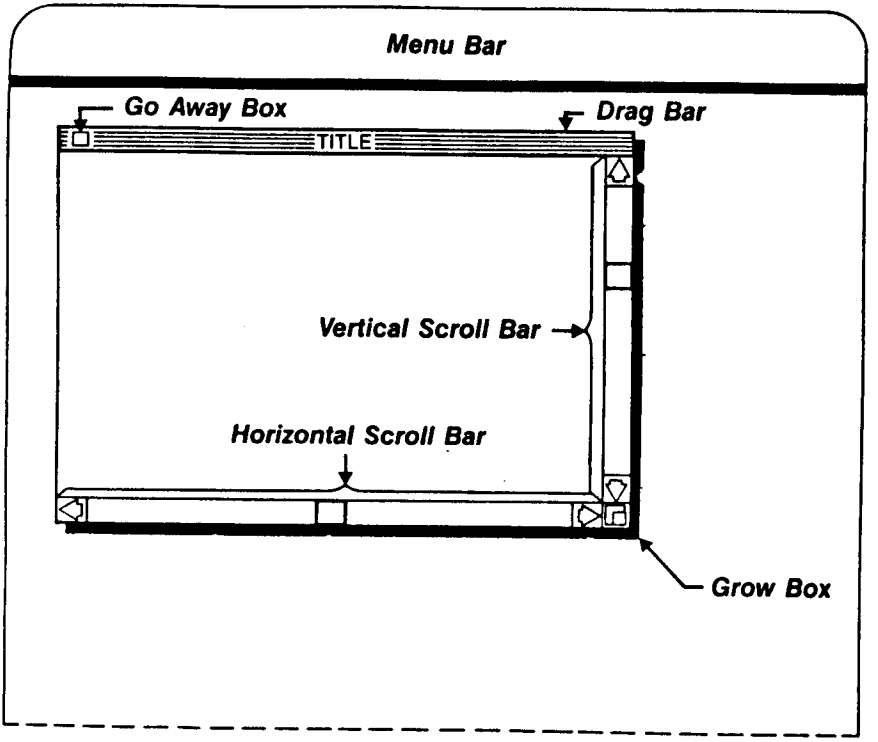
Drag Bar

The drag bar is used to move the window around on the display. To move the window, the user first positions the cursor on the drag bar and depresses the mouse button. Then, the user moves the cursor and "drags" the window to the desired position.

Close (Go-Away) Box

The drag bar also contains the window Close (or Go-Away) Box. To close the window, the user positions the cursor on the Close Box and clicks the mouse button.

Figure F-1. Components of a MouseText window.



Grow Box

The lower-right corner of the window contains the Grow Box which is used to change the size of a window. To do this, the user positions the cursor on the Grow Box and depresses (and holds down) the mouse button. The user then moves the cursor to shrink or enlarge the window. The display shows the new size of the window as an outline which moves around as the mouse moves. When the user releases the mouse button, the Toolkit redisplays the window with its new size but without contents. The program puts appropriate text into the re-sized window by calling window commands or its own window subroutines.

Window ID Numbers

Each open window must have a unique ID number within the range of 1 through 255. An attempt to open a second window with the same ID number as an already opened window will return an error.

A window ID number of 0 is not valid because the FrontWindow command returns ID = 0 when a window is not opened. An attempt to open a window with an ID number of 0 will return an error.

With some of the Toolkit commands, you can use ID number = 0 to indicate the front window. If there is no front window, these commands will return an error. The commands which interpret ID = 0 to equal to the front window, are:

- o CloseWindow
- o SelectWindow
- o Grow Window
- o Screen To Window
- o WinString
- o WinBlock
- o GetWinPtr
- o DragWindow
- o WindowToScreen
- o WinChar
- o WinText
- o WinOp

Note: The use of ID = 0 to select the front window is only a convenience. You can use the actual ID number of the front window instead.

Window Coordinate Systems

Three different coordinate systems are used with window commands.

Coordinate System	X - Range	Y - Range
Mouse coordinates	0 to 79	0 to 23
Screen coordinates	-80 to 159	-24 to 47
Window coordinates	-80 to 159	-24 to 47

The mouse coordinates correspond to the absolute range of the display screen and are expressed as unsigned byte quantities. The window and screen coordinates are represented as two-byte signed quantities.

It is important to be aware of the ranges of the signed two-byte quantities because the Toolkit routines make certain assumptions about the high byte. The only time the high byte is not simply the sign extension of the low byte's bit is when the value is in the range 128 to 159 for the X-axis. The Y-axis quantities are also two-byte quantities for the sake of consistency. The only legal values of the high byte are $00 and $FF.

To be visible, characters must be in the top window, and their screen coordinates must be in the range from 0 to 79 in the X-axis and 0 to 23 in the Y-axis. What's more, if the width of the window is W and the length of the window is L, characters are visible only if their window coordinates are in the range from 0 to W - 1 in the X-axis and 0 to L - 1 in the Y-axis.

The scroll bars are considered to be in the content area. Thus, if the vertical scroll bar is used, the useful content area range on the X-axis is from 0 to W - 3. Similarly, if there is a horizontal scroll bar, the useful content area range on the Y-axis is from 0 to L - 2.

Note: If a Grow Box is present, the vertical scroll bar space is used even if the scroll bar is not present. This ensures that the useful content area is always rectangular.

There must be at least one character in the window's content area for a Window Information Data Structure to be displayed correctly. The window length must be at least one, or 2 two if there is a horizontal scroll bar. Window width must be at least one, or three if there is a vertical scroll bar or a Grow Box. The maximum window width is 80. The maximum length is 22 for normal windows, 23 for dialog windows.

Note: It is a good idea to keep window width greater than 3. Otherwise, you may have a window whose title does not show or a window that cannot be dragged. In this situation, the window could only be closed, since there is only space for the Close Box.

A window can be placed in any position on the screen, including a position that makes part of the window invisible. This explains the ranges of the screen and window coordinates. Even though the ranges normally used are positive, you can get meaningful negative values when you convert from one coordinate system to another. For

example, a window's drag bar is always in the negative range of the window's Y-axis.

Note: Windows are output-only devices. The Toolkit will not copy their contents into user memory. The application program must ensure that the information in the content memory area and the contents of the window agree.

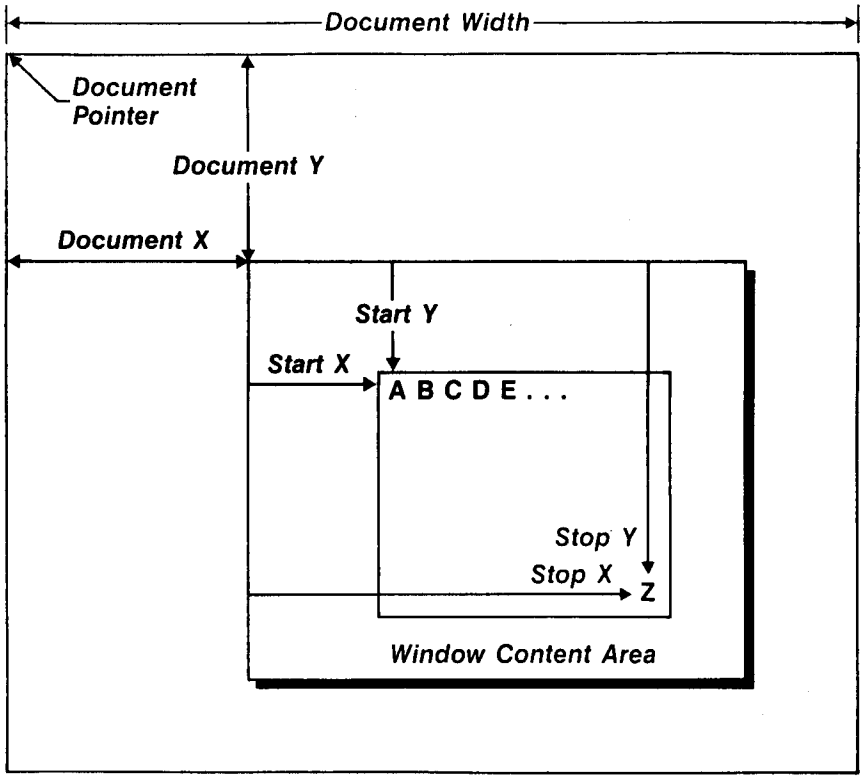
Window/Document Information

The only document display feature built into the Toolkit is a screen image of the text. Each line is padded with spaces on the right, and there are no special line delimiters. In addition, the number of characters per line is fixed.

To support the document display, the window management part of the Toolkit needs certain information about the document. This information is in the Document Information Data Structure (Dinfo) as described in Section I (Pascal Data Structures). The location of the window in the document is specified by Dinfo quantities Dx and Dy (see Figure F-2). The window can be placed anywhere within the document. In this sense, the document dimensions can be considered as a fourth coordinate system in which the window coordinates are embedded.

Other kinds of document displays are possible, but the routines to create them must be provided by the application program. For information about adding display routines, see the command "SetUserHook".

Figure F-2. Location Parameters in a Document



Refreshing Windows

Whenever a window is dragged, the Toolkit must redisplay the content areas of the windows. The application program can override the Toolkit's document display feature by having a routine that is called by the Toolkit whenever the window is to be redisplayed. The program passes the address of this assembly language routine to the Toolkit as part of the Window Information Data Structure. Because of the way the Toolkit saves zero-page locations, the program's routine cannot rely on the contents of those locations. Furthermore, the routine can only call the Toolkit's window update commands to update the content region. These commands are WinChar, WinString, WinBlock, and WinOp. (Note: WinBlock uses a Document Information Data Structure.)

In the case where the window should not be refreshed automatically, the Toolkit uses a type of event called an update event to signal the application that the window needs to be refreshed. The application specifies that a window is of this type by making the two-byte DInfo pointer (in the Window Data Structure) equal to zero. Please see the OpenWindow and GetEvent command descriptions for more information.

G. Control Region Commands

Overview

These commands deal with the control regions in the front window -- the horizontal and vertical scroll bars and the Thumbs.

Control Region Command List

No.	Name	Description
39	FindControl	Returns whether the mouse is in a control region.
40	SetCtlMax	Sets the range of a scroll bar.
41	TrackThumb	Tracks the Thumb until the mouse button is released.
42	UpdateThumb	Displays the Thumb in given position.
43	ActivateCtl	Changes the state of a scroll bar (active or inactive)

Scroll Bars

Scroll bars are the only window control regions supported by the Toolkit. The scroll bars are displayed in the content region of the front (active) window. The horizontal and vertical scroll bars may be present individually, or both may be present.

An active scroll bar has several components, as shown in Figure G-1

- o scroll arrows at both ends of the scroll bar
- o an open box called the Thumb
- o gray regions between the arrows called:
  - Page-Up and Page-Down Regions in a vertical scroll bar
  - Page-Left and Page-Right Regions in a horizontal scroll bar.

An application program should provide for three different ways of scrolling the window contents using the scroll bars.

1. Pressing the mouse button with the cursor on top of a scroll arrow. This will continuously scroll the document as long as the button is held down. During scrolling, the thumb moves to indicate the relative position of the window in the document.

2. Positioning the cursor on the Thumb, pressing the mouse button, and dragging the Thumb. This will scroll the document at an accelerated rate.

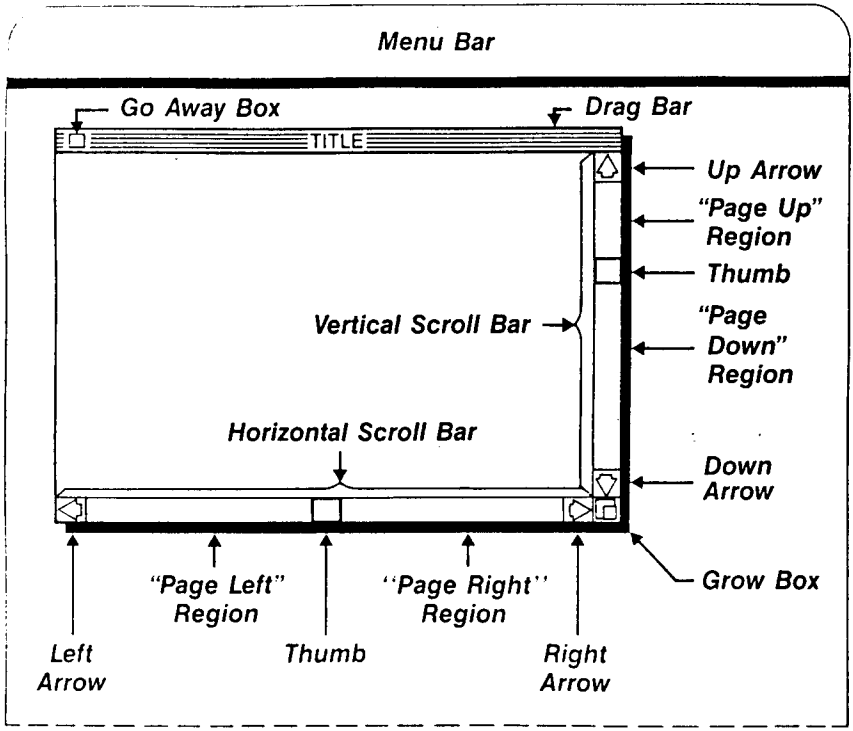
3. Pressing the mouse button with the cursor in a Page-Up or Page-Down Region. This will scroll the document up or down a full page (or window) at a time.

The Thumb should appear in the full up or full down position only when the first or last character of the document appears in the window. This ensures that the user can always page up or down, and that the Thumb can be used to get the first and last characters of a document.

If the full width or length of a document appears in the window, the scroll bars should reflect this condition by appearing in the inactive state on the display.

If the window is so narrow that less than three character cells are available for the page regions and the Thumb, the Toolkit will not display the Thumb. If fewer than than three cells are available for the entire scroll bar, not even the arrows will be shown and the user will be unable to scroll. Instead, the Toolkit will display a gray region if the scroll bar is active, or a hollow region if it is inactive.

Figure G-1. Window Control Regions.



This page left blank for your notes.

H. Technical Reference Guide

Table H -1 lists all of the Toolkit commands by name and function. Complete descriptions of the commands follow in alphabetical order. For the commands as called in the different languages, please see Section B which describes the appropriate language interface.

Table H-1. Alphabetical List of Toolkit Commands

Name	Number	Type
ActivateCtl.	43	Control Region Commands
Bload	---	Pascal Utility Procedure
CheckEvents	5	Event-Handling Commands
CheckItem	16	Menu Commands
CloseAll	25	Window Commands
Close Window	24	Window Commands
DisableItem	15	Menu Commands
DisableMenu	14	Menu Commands
DragWindow	30	Window Commands
Exit	---	Pascal Utility Procedure
FindControl	39	Control Region Commands
FindWindow	26	Window Commands
FlushEvents	07	Event-Handling Commands
FrontWindow	27	Window Commands
GetEvent	6	Event-Handling Commands
GetMachID	---	Pascal Utility Procedure
GetWinPtr	45	Window Commands
GrowWindow	31	Window Commands
HideCursor	4	Cursor Commands
HiLiteMenu	13	Menu Commands
InitMenu	9	Menu Commands
Init WindowMgr	22	Window Commands
KeyboardMouse	48	Startup Commands
Load	---	Assembler Utility Procedure
MenuKey.	12	Menu Commands
MenuSelect	11	Menu Commands
ObscureCursor	44	Cursor Command

Table H-1. Alphabetical List of Toolkit Commands (cont.)

Name	Number	Type
OpenWindow	23	Window Commands
PascIntAdr	17	Startup Commands
PeekEvent	21	Event-Handling Commands
PostEvent	46	Event-Handling Commands
ScreenWindow.	33	Window Commands
SelectWindow	28	Window Commands
SetCtlMax	40	Control Region Command
SetCursor.	2	Cursor Commands
SetKeyEvent	8	Event-Handling Commands
SetMark	20	Menu Commands
SetMenu.	10	Menu Commands
SetUserHook	47	Startup Commands
ShowCursor	3	Cursor Commands
StartDeskTop	0	Startup Commands
StopDeskTop	1	Startup Commands
Tokniz	---	Assembler Utility Procedure
TrackGoAway	9	Window Commands
TrackThumb	41	Control Region Commands
UpDateThumb	42	Control Region Commands
Version	19	Startup Commands
WinBlock	36	Window Commands
WinChar	34	Window Commands
WinOp	37	Window Commands
WindowScreen.	32.	Window Commands
WinString	35	Window Commands
WinText.	38.	Window Commands

ActivateCtl

Function: The ActivateCtl command changes the state of a scroll bar.

Command Number: 43 ($2B)

Parameter List:

2	(input, byte)	number of parameters
ctl	(input, byte)	control region to change
		0 = content region
		1 = vertical scroll bar
		2 = horizontal scroll bar
		3 = dead zone
state	(input, byte)	to make control region:
		0 = inactive
		1 = active

Description: The ActivateCtl command changes the state of a scroll bar and updates the Control Option Byte in the Winfo Data Structure. An active scroll bar shows the Thumb and page regions. An inactive bar shows a hollow page region.

The ActivateCtl command operates only on the front window.

Machine Language Commands:

```
ActivateCtl     equ     43      ; command number
actl.parms      db      2       ; parameter list for ActivateCtl
actl.ctl        db      0       ; ctl region to change
actl.inact      db      0       ; inactivate code
```

Pascal Interface:

```
Procedure ActivateCtl (whichctl: ctlarea; makeactive: boolean );
        whichctl is the control region;
                CtlArea = (NotCtl, VerScroll, HorScroll, DeadZone);
        makeactive is the control region state:
                False = inactive
                True = active
```

Error Codes:

16 ($10)	There are no windows
18 ($12)	Bad control ID (not 1 or 2)

BLOAD

Function: BLoad loads a specified "bin file.

Command Number: none

Description: The utility procedure BLoad is used to load the MouseText Runtime Module. This utility is only used with Pascal.

Pascal Interface:

```
Procedure BLoad (name: string):
        name is the name of the "bin" file to be loaded which is
                normally 'MTXKIT.ABS'.
```

Error Codes: ProDOS error codes (See Kyan Pascal User Manual)

CheckEvents

Function: CheckEvents reads the mouse, moves the cursor to the new mouse position, and posts an event, if any.

Command Number: 5 ($05)

Parameter List: 0 (input, byte) number of parameters

Description: The CheckEvents command reads the mouse and posts a mouse event if the button state has changed. The CheckEvents command posts a keypress event and clears the keyboard strobe, if a key on the keyboard is pressed and keypress events are to be checked. (Note: If a previous call to the SetKeyEvent command has disabled keypress events, the CheckEvents command ignores the keypress.) The CheckEvents command also updates the cursor position to the X and Y values of the mouse.

If the program is using Interrupt Mode, the interrupt handler calls the CheckEvent command 60 times per second, synchronized with the display vertical blanking (VBL). The Toolkit returns an error, if the program calls the CheckEvents command in Interrupt Mode.

In Passive Mode, the GetEvent command calls the CheckEvents command internally. The program should call CheckEvents or GetEvent often to ensure smooth cursor motion.

Remember: The CheckEvents command is the only command that reads the mouse and updates the cursor position. If it is never called, the cursor will never move.

An application program can have an interrupt-service routine of its own which augments or even replaces the functions of the CheckEvents command. The CheckEvents command can pass control to the routine either before or after event checking. The program can even have two interrupt routines, one called before event checking and one after. See the SetUserHook command in the "Startup Commands" for an explanation of how this is done.

CheckEvents (cont.)

If the event queue fills up, the Toolkit will ignore new events until there is room for them in the queue. To empty the queue, call the FlushEvents command.

Machine Language Commands:

```
CheckEvents   equ   5      ; command number
chke.parms    db    0      ; parameter list for CheckEvents
```

Pascal Interface: Procedure CheckEvents;

Error Codes:

7 ($07) Interrupt Mode in use. (Program specified Interrupt Mode in StartDeskTop, so it can't call CheckEvents.)

CheckItem

Function: CheckItem turns the checkmark displayed next to an item on or off.

Command Number: 16 ($10)

Parameter List:

3	(input, byte) number of parameters
id	(input, byte) menu ID
in	(input, byte) item number
ck	(input, byte) checkmark: 0 = turn checkmark off 1 = turn checkmark on

Description: The CheckItem command turns the checkmark displayed next to an item, on or off. The checkmark appears in the blank column on the left edge of the menu. Calling the CheckItem command with the item number set to 0, generates error 9 (Item Number Not Valid).

Your program can call the SetMark command to change the checkmark to any ASCII character.

Machine Language Commands:

```
CheckItem     equ   16     ; command number
chki.parms    db    3      ; parameter list for CheckItem
chki.id       db    0      ; menu ID
chki.item     db    0      ; item number
chki.chk      db    0      ; checkmark on/off
```

Pascal Interface:

```
Procedure CheckItem ( menu_id, item_num :
          integer; check : boolean );
menu_id is the menu ID number.
item_num is the item number
check is the ck (check) parameter:
          false= turn checkmark off
          true= turn checkmark on
```

Error Codes:

8 ($08) Menu ID was not found
9 ($09) Item Number not valid

CloseAll

Function: The CloseAll command closes all open windows and redisplays the screen.

Command Number: 25 ($19)

Parameter List: 0 (input, byte) number of parameters

Description: The CloseAll command removes all windows from the open window list and redisplays the screen.

Machine Language Commands:

```
CloseAll    equ   25    ; command number
cla.parms   db    0     ; parameter list for CloseAll
```

Pascal Interface: Procedure CloseAll;

Error Codes: (none)

CloseWindow

Function: The CloseWindow command removes the window with a given ID number and redisplays the screen.

Command Number: 24 ($18)

Parameter List:

1	(input, byte)	number of parameters
id	(input, byte)	ID number of window to close

Description: The CloseWindow command removes the window with a given ID number from the list of open windows, and redisplays the screen with the window removed. Setting the ID = 0 selects the top window as the window to be closed.

Machine Language Commands:

```
CloseWindow   equ   24      ; command number
cw.parms      db    1       ; parameter list for CloseWindow
cw.id         db    0       ; ID number of window to close
```

Pascal Interface: Procedure CloseWindow (window_id: integer);
window_id is the window ID number.

Error Codes:

15	($0F)	Window ID not found
17	($11)	Error returned by user hook

DisableItem

Function: The DisableItem command disables or enables selection and highlighting of menu items.

Command Number: 15 ($0F)

Parameter List:

3	(input, byte) number of parameters
id	(input, byte) menu ID
in	(input, byte) item number
dis	(input, byte) disable: 1 = disable item 0 = enable item

Description: The DisableItem command disables or enables selection and highlighting of a menu item. If an item is disabled, it cannot be selected by either the MenuSelect command or the MenuKey command. It will not be highlighted when the mouse moves to it.

To enable an item, call the DisableItem command with the disable parameter set to 0.

By setting the Disable Flag in the Menu Item Block's Item Option Byte when setting up the menu data structure, your program can make the menu item start out disabled. Afterwards, the program should use the DisableItem command to disable and enable menu items.

Calling DisableItem with item number set to zero generates error 9, Item Number Not Valid.

Machine Language Commands:

```
DisableItem     equ     15      ; command number
ditm.parms      db      3       ; parameter list for DisableItem
ditm.id         db      0       ; menu ID
ditm.item       db      0       ; item number
ditm.dis        db      0       ; disable code
```

DisableItem (cont.)

Pascal Interface: Procedure DisableItem (menu_id, item_num : integer;disable : boolean);
menu_id is the menu ID number.
item_num is the item number.
disable is the disable parameter:
false= enable
true= disable

Error Codes:
8 ($08) Menu ID was not found
9 ($09) Item Number not valid

DisableMenu

Function: The DisableMenu command disables or enables selection and highlighting over a whole menu.

Command Number: 14 ($0E)

Parameter List:

2	(input, byte) number of parameters
id	(input, byte) menu ID
dis	(input, byte) disable: 1 = disable 0 = enable

Description: The DisableMenu command disables or enables selections and highlights over a whole menu. None of the menu items can be selected, if the menu has been disabled. The MenuSelect command and the MenuKey command cannot "undo" the disabling. The menu will still appear when the user moves the mouse to the menu title, but neither the the title nor the menu items will be highlighted.

When a call to DisableMenu enables a menu, any items that were individually disabled will remain disabled. (See the DisableItem command.)

By setting the Disable Flag in the Menu Block's Option Byte when you set-up the Menu Bar data structure, your program can make the menu start out disabled. Afterwards, the program should use the DisableMenu command to disable and enable menus.

Machine Language Commands:

```
DisableMenu   equ   14      ; command number
dism.parms    db    2       ; parameter list for DisableMenu
dism.id       db    0       ; menu ID
dism.dis      db    0       ; disable menu
```

DisableMenu (cont.)

Pascal Interface: Procedure DisableMenu (menu_id : integer;
disable : boolean);
menu_id is the menu ID number.
disable is the disable parameter:
false= enable
true= disable

Error Codes: 8 ($08) Menu ID was not found

DragWindow

Function: The DragWindow command displays the outline of the window being dragged, then redisplays it in its new position.

Command Number: 30 ($1E)

Parameter List:

3	(input, byte)	number of parameters
id	(input, byte)	ID number of window being dragged
mx	(input, byte)	X mouse coordinate of starting position
my	(input, byte)	Y mouse coordinate of starting position

Description: The DragWindow command displays the outline of the window being dragged until the user releases the mouse button. It then clears the display area previously occupied by the window and redisplays the windows from back to front.

The application program should call the DragWindow command when it detects the mouse button is down in the window's drag region. In addition to the ID number of the window, the DragWindow command also needs the mouse coordinates returned in the px and py position by the FindWIndow command. This differs from the TrackGoAway and GrowWindow commands; while the Go-Away Box and the Grow Box consist of only one character each, the drag bar consists of several characters. The mouse could be in any of them when the user starts dragging the window.

Setting ID = 0 selects the front window.

An application can also use the DragWindow command in keyboard mouse emulation mode by calling it immediately after calling the KeyboardMouse command. In this mode, the Toolkit tracks the cursor and moves the window outline while the user presses the cursor keys. The user indicates the completion of the move by pressing the RETURN key or by pressing and releasing the mouse button. Pressing the ESC key terminates the command and redisplays the window in its original position.

DragWindow (cont.)

Machine Language Commands:

```
DragWindow    equ    30      ; command number
dg.parms      db     3       ; parameter list for DragWindow
dg.id         db     0       ; window ID number
dg.x          db     0       ; x mouse coord of cursor start
dg.y          db     0       ; y mouse coord of cursor start
```

Pascal Interface:

```
Procedure DragWindow ( window_id, mousex,
                       mousey: integer );
```

window_id is the window ID number.
mousex is the mouse X coordinate.
mousey is the mouse Y coordinate.

Error Codes:

15 ($0F) Window ID not found
17 ($11) Error returned by user hook
22 ($16) Operation cannot be performed

Exit

Function: This Assembly language utility is used to return the application program to Kyan's KIX environment.

Command Number: none

Parameter List: none

Description: EXIT is used in assembly language programs only. It returns control from the application program to the KIX environment at the end of program execution. As an alternative, the programmer can use the ProDOS QUIT routine to return to the ProDOS prompt.

Machine Language Commands: none

Pascal Interface: none

Error Codes: none

FindControl

Function: The FindControl command indicates in which window control region a given point is in.

Command Number: 39 ($27)

Parameter List:

4	(input, byte) number of parameters
wx	(input, word) X window coord. of point
wy	(input, word) Y window coord. of point
ctl	(output, byte) control region point is in: 0 = content region 1 = vertical scroll bar 2 = horizontal scroll bar 3 = none of the above (dead zone)
part	(output, byte) part of region point is in: 1 = Up-Arrow of vertical scroll bar, Left-Arrow of horizontal scroll bar 2 = Down-Arrow of vertical scroll bar, Right-Arrow of horizontal scroll bar 3 =Page-up region of vert. scroll bar, Page-left region of horiz. scroll bar 4 =Page-down region vert. scroll bar, Page-right region of horiz. scroll bar 5 = Thumb of scroll bar

Description: The FindControl command indicates in which window control region a given point is in. The application program should call the FindControl command when it dtermines, by means of a call to the front window, that the mouse is in the content region of the front window. Depending on the control and part codes returned by the FindControl command, the application should take appropriate action. If the mouse is in a page-up or page-down region, or in an Up-Arrow or Down-Arrow, the application scrolls the contents of the window, then calls UpDateThumb to make the Thumb reflect the new position in the file.

The application program must make sure that the wx and wy values are converted to window coordinates before calling the FindControl command.

FindControl (cont.)

Note: This command is different from the FindWindow command, which takes mouse coordinates.

Machine Language Commands:

```
FindControl     equ     39      ; command number
findc.parms     db      4       ; parameter list for FindControl
findc.wx        dw      0       ; X window coordinate of point
findc.wy        dw      0       ; Y window coordinate of point
findc.ctl       db      0       ; control region point is in
findc.part      db      0       ; part of region point is in
```

Pascal Interface:

```
Procedure FindControl (windowx, windowy: integer; var whichctl:
            ctlarea; var whichpart: ctlpart );
    windowx is the window X coordinate.
    windowy is the window Y coordinate.
    whichctl is the control region the point is in.
            ctlarea = (NotCtl, VerScroll, HorScroll, DeadZone);
    whichpart is the part of the control region the point is in.
            ctlpart = (CtlInactive, ScrollUpLeft, ScrollDownRight,
                    PageUpLeft, PageDownRight, Thumb);
```

Error Codes: 16 ($10) There are no windows

FindWindow

Function: The FindWindow command returns the ID number of the window which contains the given point.

Command Number: 26 ($1A)

Parameter List:

4	(input, byte)	number of parameters
px	(input, byte)	X mouse-coord. of point
py	(input, byte)	Y mouse-coord. of point
type	(output, byte)	type of area point is in: 0 = Desktop 1 = Menu Bar 2 = content region 3 = drag region 4 = Grow Box 5 = Close Box
id	(output, byte)	ID number of window point is in (0 if point in desktop or menu bar).

Description: The FindWindow command returns the ID number of the window which contains the given point and returns the region type that the point is in: Menu Bar, content region, drag region, Grow Box, or Close Box. The point is specified in mouse coordinates. If the point is not in a window, the FindWindow command returns an ID number of 0 and a region type of desktop.

If the point is in the content region, the application program should call the FindControl command with window coordinates of the point to determine whether the point is in a scroll bar.

FindWindow (cont.)

Machine Language Commands:

```
FindWindow    equ    26      ; command number
fdw.parms     db     4       ; parameter list for FindWindow
fdw.x         db     0       ; X coordinate of mouse
fdw.y         db     0       ; Y coordinate of mouse
fdw.type      db     0       ; type of region mouse is in
fdw.window    db     0       ; window ID number (0 = desktop)
```

Pascal Interface:

```
Procedure FindWIndow (pointx, pointy: integer; var area: Type_area;
                      var window_id: integer );
    pointx is the X coordinate of the point.
    pointy is the Y coordinate of the point.
    area is the region type of the point
        type_area = (InDeskTop, InMenuBar, InContent,
                     InDrag, InGrow, InGoAway).
    window_id is the window ID number.
```

Error Codes: (none)

FlushEvents

Function: The FlushEvents command empties the event queue.

Command Number: 7 ($07)

Parameter List: 0 (input, byte) number of parameters

Description: The FlushEvents command empties the event queue.

Machine Language Commands:

```
FlushEvents    equ 7          ; command number
flshe.parms    db 0           ; parameter list for FlushEvents
```

Pascal Interface: Procedure FlushEvents;

Error Codes: (none)

FrontWindow

Function: The FrontWindow command returns the ID number of the front window.

Command Number: 27 ($1B)

Parameter List:

1	(input, byte)	number of parameters
id	(output, byte)	ID number of front window

Description: The FrontWindow command returns the ID number of the front, or active window. It returns an ID=0 if windows are not open.

Machine Language Commands:

```
FrontWindow   equ 27      ; command number
frtw.parms    db 1        ; parameter list for FrontWindow
frtw.id       db 0        ; ID number of front window
```

Pascal Interface:

Procedure FrontWindow (var window_id: integer);
window_id is the window ID number.

Error Codes: (none)

GetEvent

Function: The GetEvent command fetches the next event from the event queue. If there is none, the GetEvent command returns the mouse position. In Passive Mode, the GetEvent command calls the CheckEvents command.

Command Number: 6 ($06)

Parameter List:

3 (input, byte) number of parameters
et (output, byte) event type:
- 0 = no event
- 1 = button down
- 2 = button up
- 3 = key pressed
- 4 = drag event
- 5 = Apple key down
- 6 = update event

ebl (output, byte) event byte 1: X coord. or key value
eb2 (output, byte) event byte 2: Y coord. or key modifier

Description: The GetEvent command fetches the next event from the event queue so the program can respond to a key press or the mouse button. In Passive Mode, the GetEvent command calls the CheckEvent command internally to make sure the latest event gets processed.

The event-type variable is a byte which indicates what caused the event to occur. The event bytes are the X and Y mouse positions from the last call to the CheckEvents command, if the event type is 0, 1, 2, 4, or 5. The event bytes are the key and the key modifier, if the event type is 3. The high bit of the key value is 0. The key modifier values are:

0 = no modifier
1 = OPEN-APPLE pressed
2 = SOLID-APPLE pressed
3 = both Apple keys pressed

GetEvent (cont.)

The drag event (et parameter = 4) is similar to a no event, except that the mouse button is pressed. After getting a button-down event, the program should get drag events or a button-up event. If the program gets a no event while waiting for a button-up event, this indicates a mouse-up event was missed, and that you don't know what the mouse position was at that time (you only know its present position). If this happens, the program must cancel any operation that is in progress.

The Apple-key down event indicates that one of the Apple keys was down when the mouse button was pressed.

An event type 6 indicates an update event. This means that a window which cannot be automatically refreshed needs updating. The window ID is returned in ebl, the key value parameter. This event occurs only when the application has set the DInfo pointer in the Window Data Structure to zero, indicating that the window cannot be automatically refreshed.

Machine Language Commands:

```
GetEvent    equ 6          ; command number
evt.parms   db  3          ; parameter list for GetEvent
evt.type    db  0          ; the event type
evt.ebl     db  0          ; event byte 1 (x or key)
evt.eb2     db 0           ; event byte 2 (y or modifier)
evt.x       equ  evt.eb1   ; x pos of mouse
evt.y       equ  evt.eb2   ; y pos of mouse
evt.key     equ  evt.eb1   ; key input by user
evt.keymod  equ  evt.eb2   ; modifier to key input by user
```

GetEvent (cont.)

Pascal Interface:

```
Procedure GetEvent ( var event : type_event );
        event is returned with:
          evt_kind set to the event type.
                char1 set to event byte 1:  X coordinate or key value.
                char2 set to event byte 2:  Y coordinate or key modifier.
        Type_event = Record
                        Evtkind: byte;
                        Char1: byte;
                        Char2: byte;
                        reserve: byte;
                      end;
        Eventkind
                0 = No event            1 = button down
                2 = Button up           3 = key pressed
                4 = drag event          5 = Apple Key down
                6 = Update event
```

Error Codes: (none)

GetMachID

Function: The GetMachID procedure gets the machine ID byte and subsidiary ID byte.

Command Number: none

Parameter List: none

Description: The utility procedure GetMachID reads the machine ID byte at $FBB3 and the subsidiary ID byte at $FBC0. It returns the values in the calling parameters. The ID byte is $06 for Apple IIe and Apple IIc. The subsidiary ID byte is:

$EA = Apple IIe
$E0 = Apple IIe with enhanced ROM
$00 = Apple IIc.

This procedure is only used with Pascal.

Machine Language Commands: Not applicable

Pascal Interface:

Procedure GetMachID (Var machid, machsid: integer);

Get MachID is returned with:
machid is the ID byte.
machsid is the subsidiary ID byte

Error Codes: none

GetWinPtr

Function: TheGetWinPtr command returns the pointer to the Winfo structure of the open window that has a specified ID number.

Command Number: 45 ($2D)

Parameter List:
2 (input, byte) number of parameters
id (input, byte) ID number of window
ptr (output, word) pointer to Winfo Data Structure

Description: The GetWinPtr command returns the pointer to the Window Information Data Structure (Winfo) of the open window which has the specified ID number. Setting ID = 0 selects the top window.

Machine Language Commands:

```
GetWinPtr    equ   45     ; command number
gwip.parms   db    2      ; parameter list for GetWinPtr
gwip.id      db    0      ; window ID number
gwip.winfo   dw    0      ; pointer to Winfo Data Structure
```

Pascal Interface:
Procedure GetWinPtr (window_id: integer; var MyWinfoPtr: winfo_ptr);
window_id is the ID number of the window.
winfo_ptr is a pointer to Winfo data structure.
Winfo_Ptr = ^Winfo;

Error Codes: 15 ($0F) Window ID not found

GrowWindow

Function: The GrowWindow command displays the outline of the window being grown, then redisplays an empty window with the new size.

Command Number: 31 ($1F)

Parameter List:

	1	(input, byte) number of parameters
	stat	(output, byte) return status: 0 = window did not change size 1 = window did change size

Description: The Grow Window command displays the outline of the window being grown until the user releases the mouse button. The GrowWindow command then clears the display area previously occupied by the window and redisplays the window from the back to front.

The application program should call the GrowWindow command when it detects that the mouse button is down in the Grow Box of the front window.

The GrowWindow command leaves the content area of the front window blank because it can't determine whether the bottom of the document has been passed or whether the content area should be shifted. If the return status indicates that the GrowWindow command changed the size of the window, the application must redisplay the content area and update the scroll bars.

An application can also use the GrowWindow command in keyboard mouse emulation mode by calling it immediately after calling the KeyboardMouse command. In this mode, the Toolkit tracks the cursor and draws the window outline in different sizes while the user presses cursor keys. The user indicates the completion of the resizing by pressing the RETURN key or by pressing and releasing the mouse button. Pressing the ESC key terminates the command and redisplays the window in its original size.

GrowWindow (cont.)

Machine Language Commands:

```
GrowWindow    equ    31      ; command number
grow.parms    db     1       ; parameter list for GrowWindow
grow.resul    db     0       ; return status
```

Pascal Interface:

```
Procedure GrowWindow ( var makeitgrow:
        boolean );
makeitgrow is the return status:
        false = window did not grow
        true = window did grow
```

Error Codes:

16 ($10) There are no windows
17 ($11) Error returned by user hook
22 ($16) Operation cannot be performed

HideCursor

Function: HideCursor makes the cursor invisible.

Command Number: 4 ($04)

Parameter List: 0 (input, byte) the number of parameters

Description: The HideCursor command makes the cursor invisible. ShowCursor has no effect if the cursor is temporarily invisible.

Machine Language Commands:

```
HideCursor    equ 4          ; command number
hidec.parms   db 0           ; parameter list for HideCursor
```

Pascal Interface: Procedure HideCursor;

Error Codes: (none)

HiLiteMenu

Function: The HiLiteMenu command turns menu title highlighting on and off.

Command Number: 13 ($0D)

Parameter List: 1 (input, byte) number of parameters
id (input, byte) menu ID: 0 = turn highlight off

Description: The HiLiteMenu command highlights specified menu titles in the Menu Bar. To turn off highlighting after a call to the MenuSelect command or MenuKey command, call the HiliteMenu command with id = 0.

Machine Language Commands:

```
HiLiteMenu    equ 13        ; command number
hili.parms    db 1          ; parameter list for HiLiteMenu
hili.mid      db 0          ; menu ID (0 for all)
```

Pascal Interface: Procedure HiLiteMenu (menu_id : integer);
menu_id is the menu number.

Error Codes: 8 ($08) Menu ID was not found.

InitMenu

Function: The InitMenu command establishes an area of memory which is used to save the part of the display obscured by menus.

Command Number: 9 ($09)

Parameter List:

2	(input, byte) numbers of parameters
sa	(input, word) save area: pointer to reserved memory area
sas	(input, word) save area size: number of bytes reserved

Description: During calls to the MenuSelect command, the part of the display obscured by a menu must be saved so that it can be replaced when the menu goes away. The application program must provide memory space and reserve it for the Toolkit.

You can determine the amount of memory space to reserve for menu displays by calculating the screen area of the largest menu in the program. The largest menu could have a large screen area, or it could have only a few items, each of which is very long.

You calculate the screen area of a menu by taking the product of the number of items in the menu, plus 1, and multiplying it by five bytes more than the length of the longest item string in that menu. If you are using keys to select items, each item string must include three bytes for displaying a space, an Apple icon, and the key which selects the item.

When the program calls the SetMenu command to initialize a menu bar, the SetMenu command checks whether the amount of memory reserved by the InitMenu command is sufficient for a particular menu and returns an error if it is not.

InitMenu (cont.)

Machine Language Commands:

```
InitMenu    equ 9          ; command number
im.parms    db 2           ; parameter list for InitMenu
im.sarea    dw savearea    ; area used for saving screen under
                             menu
im.ssize    dw savesize    ; size of save area
```

Pascal Interface:

Procedure InitMenu (save: savebuffer; buf_size: integer);

savebuffer is a pointer to the save area.
buf_size is the save area size.

Error Codes: (none)

InitWindowMgr

Function: The InitWindowMgr command initializes the internal list of open windows and establishes an area of memory. This area of memory is used to save parts of the display while a window is being dragged or grown.

Command Number: 22 ($16)

Parameter List:

2	(input, byte	number of parameters
ptr	(input, word)	pointer to reserved memory area
size	(input, word)	size of reserved memory area in bytes

Description: The InitWindowMgr command resets the pointers to the first and last entries in the internal linked list of open windows. It alsoestablishes an area of memory which is used to save parts of the display while a window is being dragged or grown.

During calls to the DragWindow and GrowWindow commands, the Toolkit must save the part of the display obscured by the window outline so that the display can be replaced when the window operation is finished. The application program must provide the necessary memory and reserve it for use by the Toolkit.

The amount of memory space required is determined by the perimeter of the largest window (the sum of twice the window's width plus twice its length.

Note: This window memory can be the same area which is reserved by the InitMenu command.

InitWindowMgr (cont.)

Machine Language Commands:

```
InitWindowMgr  equ 22          ; command number
iwm.parms      db 2            ; parameter list for InitWindowMgr
iwm.sarea      dw savearea     ; area to use when saving window
                                 screen
iwm.ssize      dw savesize     ; size of save area
```

Pascal Interface:

```
Procedure InitWindowMgr (Var save: SaveBuffer; buf_size : integer );
        Save is the buffer.
        SaveBuffer = Array[1..SaveSize] of byte;
        buf_size is the buffer size.
```

Error Codes: (none)

KeyboardMouse

Function: The KeyboardMouse command makes the next command work in mouse emulation mode, assuming the next command is one of the three that work in that mode.

Command: 48 ($30)

Parameter List: 0 (input, byte) the number of parameters

Description: The KeyboardMouse command is a procedure call; it has no parameters.

The KeyboardMouse command is used in conjunction with the three commands that operate in mouse emulation mode: MenuSelect, DragWindow, and GrowWindow. Call the KeyboardMouse command before calling one of these commands, to make them operate in mouse emulation mode.

An application can also require this form of mouse emulation on the MenuKey command by calling the command with the ESC key as the keystroke. This has the same effect as calling the KeyboardMouse command and then calling the MenuSelect command.

Machine Language Commands:

```
KeyboardMouse    equ  48 ; command number
kdbms.parms      db 0    ; parameter list for KeyboardMouse
```

Pascal Interface: Procedure KeyboardMouse

Error Codes: (none)

Load

Function: The Load routine loads a specific "bin" file.

Command Number: none

Parameter List: none

Description: The utility assembly language routine Load is used to load the MouseText Runtime Module. This routine is used with assembly language programs only.

Machine Language Commands:

Put lsb of filename string in the x register
Put msb of filename string in the y register
jump to subroutine load

Pascal Interface: none

Error Codes: ProDOS error calls (see Kyan Pascal User Manual)

MenuKey

Function: The MenuKey command finds the menu item which matches a key.

Command Number: 12 ($0C)

Parameter List:

4 (input, byte) number of parameters
id (output, byte) menu ID, 0 if no item selected
in(output, byte) item number, undefined if id=0
k (input, byte) key: the character typed
km (input, byte) key modifier, as returned by GetEvent:
- 0 = no modifier,
- 1 = OPEN-APPLE key
- 2 = SOLID-APPLE key
- 3 = either Apple key

Description: After the user presses a key, the MenuKey command searches the menu data to find a menu item that has a matching key. If it finds a match, it highlights the menu title and returns the menu ID number and item number. This is also what the MenuSelect command does. In addition, the MenuKey acts like the MenuSelect command and leaves the selected menu highlighted. The program must call the HiliteMenu command to turn off the highlighting.

If you set the key modifier parameter to 3, either Apple key will serve to modify a matching keypress,

If an item is disabled, its menu key or keys will not select it.

As a special case, the MenuKey command can operate like the MenuSelect command in keyboard mouse emulation mode. Calling the MenuKey command with ESC as the key, initiates this mode of operation. The Toolkit tracks the cursor while the user presses the cursor keys. Selections are made by pressing the RETURN key or pressing and releasing the mouse button. The user can also press an appropriate command key. Pressing the ESC terminates the command.

MenuKey (cont.)

Machine Language Commands:

```
MenuKey     equ 12          ; command number
mkey.parms  db 4            ; parameter list for MenuKey
mkey.mid    db 0            ; menu ID returned
mkey.item   db 0            ; item number returned
mkey.key    db 0            ; key user typed
mkey.mod    db 0            ; modifier of key
```

Pascal Interface:

```
Procedure MenuKey ( var menu_id, menu_choice : integer; var
                        key_event : type_event );
    menu_id is the menu ID number.
    menu_choice is the item number.
    key_event is returned with:
        char1 as the key value
        char2 as the key modifier
    Type_event = Record
                    EvtKind: byte;
                    Char1 : byte;
                    Char2 : byte;
                    Reserve: byte;
                 end;
```

Error Codes: (none)

MenuSelect

Function: The MenuSelect command interacts with the mouse to display a menu and return the selection, if any.

Command Number: 11 ($0B)

Parameter List:
2 (input, byte) number of parameters
id (output, byte) menu ID, 0 = no menu item chosen
in (output, byte) menu item number, undefined if id = 0

Description: The MenuSelect command performs the interactive display of menus which occurs while the user keeps the mouse button depressed. MenuSelect does not return until the user releases the button and a button-up event occurs.

The application program calls the MenuSelect command whenever the user presses the mouse button on line 0 of the display. As the user moves the mouse up and down the menu display, the MenuSelect command tracks the mouse and updates the cursor. When the cursor moves to a menu item, the MenuSelect command highlights the item.

When the user releases the mouse button with the cursor on a menu item, the MenuSelect command removes the menu from the display, highlights the menu title, and returns the menu ID number and the item number. When the program finishes performing the selected operation, it must call the HiLiteMenu command to turn off the highlighted portion of the menu title.

An application can also use the MenuSelect command in keyboard mouse emulation mode. By calling it immediately after calling the KeyboardMouse command, the Toolkit tracks the cursor while the user presses cursor keys to move the cursor. The user selects a menu item by pressing the RETURN key or by pressing and releasing the mouse button. The user can also press an appropriate command key. Pressing the ESC key terminates the command.

MenuSelect (cont.)

Machine Language Commands:

```
MenuSelect    equ 11        ; command number
ms.parms      db 2          ; parameter list for MenuSelect
ms.mid        db 0          ; menu ID returned
ms.item       db 0          ; item number returned
```

Pascal Interface:

Procedure MenuSelect (var menu_id, menu_choice : integer);
menu_id is the menu ID number.
menu_choice is the menu item number.

Error Codes: (none)

ObscureCursor

Function: ObscureCursor makes the cursor temporarily invisible.

Command Number: 44 ($2C)

Parameter List: 0 (input, byte) the number of parameters

Description: The ObscureCursor command makes the cursor invisible until the mouse moves, then it reappears. Use ObscureCursor when text is being entered, and you do not want to obscure the view of the text. As soon as you move the mouse to perform another task, the cursor reappears.

Machine Language Commands:

```
ObscureCursor equ 44          ; command number
obscc.parms   db 0            ; parameter list for ObscureCursor
```

Pascal Interface: Procedure ObscureCursor;

Error Codes: (none)

OpenWindow

Function: The OpenWindow command opens a window by supplying a pointer to the window's Information Data Structure.

Command Number: 23 ($17)

Parameter List:

1	(input, byte) number of parameters
ptr	(input, word) pointer to WINFO Data Structure

Description: The OpenWindow command passes window information to the Toolkit via a pointer to the Window Information Data Structure, or the Winfo Data Structure (See Table H-1). The Winfo Data Structure must reside in a fixed location in memory while the window is open.

The Window Information Data Structure includes a pointer to the Document Information Data Structure (Dinfo Data Structure) which the Toolkit uses to obtain display text for the window (See Table H-5). Each call to the OpenWindow command makes that particular window the front, or active window.

The OpenWindow command forces X and Y position coordinates into valid values. The command also forces the Thumb positions to be no larger than the maximum size. However, the OpenWindow command does not check to insure that window minimums are smaller than maximums or that the current window size is between the maximum and minimum scale.

The application program can substitute its own routine for the OpenWindow command. The program passes the address of its open routine in the Winfo Data Structure to the pointer in the Dinfo Data Structure and sets bit 7 of the Window Option Byte. The Toolkit will pass control to the program's routine whenever the contents of the window need to be changed.

OpenWindow (cont.)

Because the user routine is called from within the Toolkit, it cannot rely on the zero-page locations the Toolkit uses. They are currently $00 to $18. When the Toolkit calls the user routine, the register contains the following values:

- o accumulator: window ID number
- o X register: low byte of Winfo address
- o Y register: high byte of Winfo address

The routine can only call Toolkit commands with names which start with Win-. These commands will update the window content region it was requested to update. Any other calls can put the Toolkit into an unknown state.

OpenWindow (cont.)

Table H-1. Information Structure for a Window

ParameterFunction	Size	Note
Window ID Number (not 0)	1 byte	
Window Option Byte	1 byte	
Title String Pointer	2 bytes	
Window Position X Coordinate	2 bytes	1, 2
Window Position Y Coordinate	2 bytes	1, 2
Current Content Width	1 byte	1, 3
Current Content Length	1 byte	1, 3
Minimum Content Width	1 byte	
Maximum Content Width	1 byte	4
Minimum Content Length	1 byte	
Maximum Content Length	1 byte	4
Document Information Structure Pointer	2 bytes	7
Horizontal Control Option Byte	1 byte	
Vertical Control Option Byte	1 byte	
Horizontal Scroll Maximum	1 byte	
Current Horizontal Thumb Position	1 byte	1, 5
Vertical Scroll Maximum	1 byte	
Current Vertical Thumb Position	1 byte	1, 5
Window Status Byte	1 byte	1
Reserved for Future Use	1 byte	6
Pointer to Next Winfo Structure	2 bytes	6
Reserved for Toolkit	2 bytes	6
Screen Area Covered	4 bytes	6

1 Program sets initial values, Toolkit updates these.
2 Initial values determine initial position of window.
3 Initial values determine initial window size.
4 Document width & length determine max content width & length.
5 Initial values determine initial position of thumb.
6 Items filled by Toolkit.
7 If the pointer is zero, the next GetEvent will signal & Update_Event

OpenWindow (cont.)

Table H-2. Contents of Window Option Byte in Window Information Structure

Bit No.	Function	Notes
7	Document Pointer Function	1
6, 5	Reserved for Future Use	
4, 3	Reserved for Toolkit	2
2	Grow Box is present	3
1	Close Box is present	3
0	Window is Dialog or Alert Box	3

1 This bit indicates the function of the Document Pointer.
0 = Pointer to Document Information Structure
1 = Pointer to User Window Routine

2 The program must set these bytes to 0.

3 These items set the initial appearance of the window. They cannot be changed when the window is open. You must close the window, change the values, then open the window again, if you want to change them.

Table H-3. Contents of Horizontal or Vertical Control Option Byte in Window Information Structure

Bit No.	Function	Notes
7	Scrollbar is present	1
6	Thumb is present	1
5 - 1	Reserved for Future Use	
0	Scrollbar is active	2

1 These items set the initial appearance of the window. They cannot be changed when the window is open. You must close the window, change the values, then open the window again, if you want to change them.

2 Initial value set by program. Afterwards, use ActivateCtl to change it.

OpenWindow (cont.)

Table H-4. Contents of Window Status Byte in Window Information Structure

Bit No.	Function	Note
7	Window is open	1
6 - 4	Reserved for Future Use	
3 - 0	Used by Toolkit	

1 Program can read to determine state of window.

Table H-5. Information Structure for a Document

Parameter Function	Size	Note
Document Pointer	2 bytes	1
Reserved (set to 0)	1 byte	
Document Width	1 byte	
Document X Coordinate	2 bytes	2
Document Y Coordinate	2 bytes	2
Reserved for Toolkit	4 bytes	3

1 See bit 7 of the Window Option Byte.
2 Set to 0 or set initial position in the document.
3 The program must set these bytes to 0.

Machine Language Commands:

```
OpenWindow   equ 23        ; command number
open.parm    db 1          ; parameter list for OpenWindow
open.wind    dw 0          ; pointer to Winfo Data Structure
```

OpenWindow (cont.)

Pascal Interface:

```
Procedure OpenWindow (var my_Winfo:winfo);
        my_Winfo is the Winfo data structure.
          Winfo = Record
                        WindowID: byte;
                        WinOpt: byte;
                        TitlePtr: ^TitleStr;
                        WindowX: integer;
                        WindowY: integer;
                        ContWidth: byte;
                        ContLength: byte;
                        MinContWidth: byte;
                        MaxContWidth: byte;
                        MinContLength: byte;
                        MaxContLength: byte;
                        DinfoPtr: ^Dinfo;
                        HorContOpt: byte;
                        VertContOpt: byte;
                        HThumbMax: byte;
                        HThumbPos: byte;
                        VThumbMax: byte;
                        VThumbPos: byte;
                        WinStatus: byte;
                        Reserve1: byte;
                        NextWinfo: ^Winfo;
                        Reserve2: byte;
                        Reserve3: byte;
                        Reserve4: byte;
                        Reserve5: byte;
                        Reserve6: byte;
                        Reserve7: byte;
                end;
        Dinfo_ptr = ^Dinfo;
```

OpenWindow (cont.)

```
DInfo = Record
            DocPtr: integer;
            Reserve1: byte;
            DocWidth: byte;
            DocX: integer;
            DocY: integer;
            DocLength: integer;
            Reserve2: integer;
            Reserve3: integer;
        end;
```

Error Codes:

12 ($0C)	A window with same ID is already open
13 ($0D)	InitWindowMgr buffer too small for this window
14 ($0E)	Bad WInfo--tried to open with ID=0, or conflicting max and min width or length
17 ($11)	Error returned by user hook

PascIntAdr

Function: PascIntAdr returns the address of the Toolkit's interrupt handler.

Command Number: 17 ($11)

Parameter List:

1	(input, byte)	the number of parameters
Adr	(output, word)	the address of the interrupt handler

Description: The PascIntAdr command returns the address of the Toolkit's interrupt handler in the Adr parameter. Your Pascal program can pass that address on to the Mouse Attach Driver (see Appendix II) when it calls SetMouse. The SetMouse call should always specify Passive Mode along with the interrupt address. The program should do this before calling StartDeskTop, which will enable interrupts if its Int parameter is set to 1.

Note: This command is used only in Pascal programs.

Machine Language Commands:

```
PascIntAdr      equ 17          ; command number
pasc.parms      db 1            ; parameter list for PascIntAdr
pasc.addr       dw 0            ; address of int handler
```

Pascal Interface: Procedure PascIntAdr (var IntAdr: integer);
IntAdr is address of interrupt routine

Error Codes: (none)

PeekEvent

Function: The PeekEvent command reports on next event actions without removing them from the queue.

Command Number: 21 ($15)

Parameter List:

3	(input, byte)	number of parameters
et	(output, byte)	event type: 0 = no event 1 = button down 2 = button up 3 = key pressed 4 = drag event 5 = Apple key down 6 = update event
eb1	(output, byte)	event byte 1: X coordinate or key value
eb2	(output, byte)	event byte 2: Y coordinate or key modifier

Description: The PeekEvent command returns information from next event actions in the event queue, but does not remove them from the queue. The parameters are the same as for the GetEvent command, described earlier.

Machine Language Commands:

```
PeekEvent     equ  21          ; command number

pke.parms     db 3             ; parameter list for PeekEvent
pke.type      db 0             ; the event type
pke.eb1       db 0             ; event byte 1 (x or key)
pke.eb2       db 0             ; event byte 2 (y or modifer)
pke.x         equ  pke.eb1     ; x pos of mouse
pke.y         equ  pke.eb2     ; y pos of mouse
pke.key       equ  pke.eb1     ; key input by user
pke.keymod    equ  pke.eb2     ; modifier to key input by user
```

PeekEvent (cont.)

Pascal Interface:

```
Procedure PeekEvent ( var event : type_event );
        event is returned with:
                evt_kind set to the event type.
                char1 set to event byte 1:  X coordinate or key value.
                char2 set to event byte 2:  Y coordinate or key modifier.
        Type_event = record
                        EvtKind: byte;
                        Char1: byte;
                        Char2: byte;
                        reserved: byte;
                       end;
```

Error Codes: (none)

PostEvent

Function: The PostEvent command posts an event into the event queue.

Command Number: 46 ($2E)

Parameter List:

3	(input, byte)	number of parameters
et	(input, byte)	event type: 0 = no event 1 = button down 2 = button up 3 = key pressed 4 = drag event 5 = Apple key down 6 = update event
eb1	(input, byte)	event byte 1: X coordinate or key value
eb2	(input, byte)	event byte 2 Y coordinate or key modifier

Description: The PostEvent command posts an event into the event queue. The parameter list is the same for the GetEvent command except that all of the parameters are inputs.

The PostEvent command can have an event type similar to the one returned by the GetEvent command (et = 0, 1, ...5). It can also have a type defined by the application program (et = 128, 129, ...255). Any other value for the et parameter is illegal. The Toolkit ignores events 128-255.

Machine Language Commands:

```
PostEvent     equ 46        ; command number
post.parms    db 3          ; parameter list for PostEvent
post.type     db 0          ; the event letter
post.eb1      db 0          ; event byte 1 (x or key)
post.eb2      db 0          ; event byte 2 (y or modifier)
```

PostEvent (cont.)

```
post.x          equ  post.eb1   ; x pos of mouse
post.y          equ  post.eb2   ; y pos of mouse
post.key        equ  post.eb1   ; key input by user
post.keymod     equ  post.eb2   ; modifier to key input by user
```

Pascal Interface:

```
Procedure PostEvent ( var event : type_event );
        event should be supplied with:
           evt_kind set to the event type.
                char1 set to event byte 1:  X coordinate or key value.
                char2 set to event byte 2:  Y coordinate or key modifier.
        Type_event = Record
                        Evtkind: byte;
                        Char1: byte;
                        Char2: byte;
                        Reserved: byte;
                       end;
```

Error Codes: 19 ($13) The event queue is full; event not posted.
20 ($14) Illegal event type; event not posted.

ScreenWindow

Function: The ScreenWindow command converts screen coordinate values to window coordinates.

Command Number: 33 ($21)

Parameter List:

5	(input, byte)	number of parameters
id	(input, byte)	ID number of window to use
sx	(input, word)	X coordinate for the screen
sy	(input, word)	Y coordinate for the screen
wx	(output, word)	X coordinate in the window
wy	(output, word)	Y coordinate in the window

Description: The ScreenToWindow command converts passed coordinate values from screen coordinates to window coordinates. Setting ID = 0 selects the front window.

Machine Language Commands:

```
ScreenWindow  equ 33      ; command number
s2w.parms     db 5        ; parameter list for ScreenWindow
s2w.id        db 0        ; window ID number
s2w.sx        dw 0        ; X screen coordinate
s2w.sy        dw 0        ; Y screen coordinate
s2w.wx        dw 0        ; X coordinate in window
s2w.wy        dw 0        ; Y coordinate in window
```

Pascal Interface

```
Procedure ScreenWindow ( window_id, screenx, screeny: integer;
                         var windowx, windowy: integer );
```

window_id is the window ID number.
screenx is the screen X coordinate.
screeny is the screen Y coordinate.
windowx is the window X coordinate.
windowy is the window Y coordinate.

Error Codes: 15 ($0F) Window ID not found

SelectWindow

Function: The SelectWindow command activates the window with a given ID number.

Command: 28 ($1C)

Parameter List:	1	(input, byte)	number of parameters
	id	(input, byte)	ID number of window

Description: The SelectWindow command makes the window with the given ID number the front, or active, window and redisplays the screen. The window which was active becomes the second window in the list. Setting ID = 0 selects the front window. If the window selected is already the front window, the Toolkit does not redisplay the screen.

Machine Language Commands:

```
SelectWindow  equ 28        ; command number

selw.parms    db 1          ; parameter list for SelectWindow
selw.id       db 0          ; ID number of window
```

Pascal Interface

```
Procedure SelectWindow ( window_id: integer );
        window_id is the window ID number.
```

Error Codes:	15 ($0F)	Window ID not found
	17 ($11)	Error returned by user hook

SetCtlMax

Function: The SetCtlMax command changes the range of the scroll bar of the front window.

Command Number: 40 ($28)

Parameter List:

2	(input, byte)	number of parameters
ctl	(input, byte)	control region to update max value for: 1 = vertical scroll bar 2 = horizontal scroll bar
max	(input, byte)	new maximum value (must be >1)

Description: The SetCtlMax command changes the range of the scroll bar of the front window. If the current Thumb position is greater than the new maximum, the SetCtlMax command sets the Thumb to the new maximum and calls the UpdateThumb command to display it in the proper position. The SetCtlMax command changes the control max value and, if necessary the Thumb position in the Winfo Data Structure.

The program normally calls the SetCtlMax command whenever the size of a window changes.

Maximum values depend on the application. A typical maximum value for the horizontal scroll bar would be calculated as the document width, minus the content width, plus twice the width of the vertical scroll bar or grow box.

A typical maximum value for the vertical scroll bar would be calculated as the document length, minus the content length, plus the height of the horizontal scroll bar.

Machine Language Commands:

```
SetCtlMax       equ  40         ; command number
setct.parms     db   2          ; parameter list for SetCtlMax
setct.ctl       db   0          ; control region affected
setct.newmax    db   0          ; new maximum value
```

SetCtlMax

Pascal Interface

Procedure SetCtlMax (whichctl: ctlarea; newmax: integer);

whichctl is the control region.

CtlArea = (NotCtl, VerScroll, HorScroll, DeadZone).

newmax is the new maximum value.

Error Codes:

16	($10)	There are no windows
18	($12)	Bad control ID (not 1 or 2)

SetCursor

Function: SetCursor sets the character used for displaying the cursor.

Command Number: 2 ($02)

Parameter List:

1	(input, byte)	the number of parameters
cc	(input, byte)	character to use as cursor

Description: The SetCursor command sets the character displayed as the cursor. Characters normally used as the cursor include the following MouseText characters.

Arrowhead	(ASCII value 02 $02)
Hourglass	(ASCII value 03 $03)
Checkmark	(ASCII value 04 $04)
Text Cursor	(ASCII value 20 $14)
Cell Cursor	(ASCII value 29 $1D)

If the cursor is visible, it changes to the new character as soon as SetCursor is called. Each time the cursor is moved, if it is visible, the Toolkit saves the character at the new cursor position and replaces it with the character specified by SetCursor.

Machine Language Commands:

```
SetCursor     equ 2          ; command number
setc.parms    db 1           ; parameter list for SetCursor
setc.char     db $00         ; character to use for cursor
```

Pascal Interface:

```
Procedure SetCursor ( new_ch : integer );
        new_ch is the character to use as cursor.
```

Error Codes: (none)

SetKeyEvent

Function: The SetKeyEvent command specifies whether the Toolkit treats keypresses as events.

Command Number: 8 ($08)

Parameter List:

1	(input, byte)	Number of parameters
sk	(input, byte)	set keyevent: 0 = don't check keyboard, 1 = check the keyboard

Description: The SetKeyEvent command specifies whether the Toolkit posts keypresses as events. The Toolkit reads the keyboard if the value of sk is 1. The Toolkit posts a key event and clears the key strobe, if a key is pressed. The Toolkit doesn't handle keypresses if the value of sk is 0. The Toolkit is set to post keyboard events at start up.

The Toolkit handles keypresses as events in the queue, providing a form of type-ahead.

Machine Language Commands:

```
SetKeyEvent     equ 8           ; command number
setkey.parms    db 1            ; parameter list for SetKeyEvent
setkey.sk       db 0            ; set key event
```

Pascal Interface:

```
Procedure SetKeyEvent ( chk_keyboard : boolean );
        chk_keyboard is the sk (set keyevent) parameter:
                false= don't check keyboard
                true= check the keyboard
```

Error Codes: (none)

SetMark

Function: The SetMark command enables a program to select the character to display for items which are checked in a menu.

Command Number: 20 ($14)

Parameter List:

4	(input, byte)	number of parameters
id	(input, byte)	menu ID
in	(input, byte)	item number
mk	(input, byte)	checkmark: 0 = use checkmark character 1 = install new character
char	(input, byte)	character to display for this item

Description: The SetMark command sets the character which is displayed when a program calls the CheckItem command. The checkmark is the default character.

Machine Language Commands:

```
SetMark      equ 20      ; command number
setm.parms   db 4        ; parameter list for SetMark
setm.id      db 0        ; menu ID
setm.item    db 0        ; item number
setm.chk     db 0        ; checkmark code
setm.char    db 0        ; character to use as checkmark
```

Pascal Interface:

```
Procedure SetMark ( menu_id, item_num: integer; mark_on: boolean;
                    mark_char : char);
```

menu_id is the menu ID number.
item_num is the menu item number.
mark_on is the mark on parameter.
mark_char is the mark char parameter.

Error Codes:

8 ($08) Menu ID was not found
9 ($09) Item Number not valid

SetMenu

Function: The SetMenu command initializes the menu bar data structure and displays the menu bar.

Command Number: 10 ($0A)

Parameter List:

1	(input, byte)	number of parameters
mbs	(input, word)	pointer to menu bar structure

Description: The SetMenu command initializes a menu bar data structure and displays the menu bar. Given a pointer to a menu bar structure (see Tables H-6 and H-8), the SetMenu command fills in the Data required by the menu commands and saves the pointer for their use. Once the SetMenu command has been called, the program must not move the data structure.

The SetMenu command checks to insure that the memory area reserved by the InitMenu command is sufficient to handle the display area which will be obscured by the menu bar. This menu bar is specified by the data structure. If the area is not sufficient, the SetMenu command returns an error, but still displays the menu bar.

SetMenu (cont.)

Table H-6. Data Structure for a Menu Bar

Parameter Function	Parameter Size
Number of Menus	1 byte
Reserved for Future Use	1 byte
First Menu Block:	
Menu ID (can't be 0)	1 byte
Menu Option Byte	1 byte
Pointer to Title String	2 bytes
Pointer to Menu Data Structure	2 bytes
X Position for Title Display	1 byte *
Left for HiLite and Select	1 byte *
Right for Hilite and Select	1 byte *
Reserved for Future Use	1 byte *
Second Menu Block:	
(same structure as First Menu Block)	
.	
.	
Last Menu Block	
(same structure as First Menu Block)	

* Indicates items filled in by Toolkit

SetMenu (cont.)

Table H-7. Contents of Option Byte in Each Menu Block

Bit Number	Bit Function
7	Disable Flag **
6	Reserved for Future Use
5	Reserved for Future Use
4	Reserved for Toolkit
3	Reserved for Toolkit
2	Reserved for Future Use
1	Reserved for Future Use
0	Reserved for Future Use

** The Disable Flag is updated by the DisableMenu command. By setting the flag in the off position before calling the SetMenu command, the program will make the menu start out disabled.

SetMenu (cont.)

Table H-8. Data Structure for a Menu

Parameter Function	Size	Note
Number of Items	1 byte	
Left Column of Save Box	1 byte	1
Right Column of Save Box	1 byte	1
Reserved for Future Use	1 byte	1
First Menu Item Block:		
Item Option Byte	1 byte	
Mark Character	1 byte	2
Character 1 (high bit off)	1 byte	3
Character 2 (high bit off)	1 byte	3
Pointer to Item String	2 bytes	
Second Menu Item Block:		
(same structure as First Menu Item Block)		
.		
.		
Last Menu Item Block:		
(same structure as First Menu Item Block		

1. Indicates items filled in by the Toolkit.
2. Updated by the SetMark command. The program can set the initial mark character in the data structure, but afterwards it should change the mar character only by calling the SetMark command.
3. The program should set this byte to 0 if not using characters.

SetMenu (cont.)

Table H-9. Contents of Option Byte in Menu Data Structure

Bit Number	Function	Notes
7	Disable Flag	1, 4
6	Item is Filler	2
5	Item is Checked	3, 4
4	Reserved for Toolkit	
3	Reserved for Toolkit	
2	Item has Mark	3, 4
1	Modifier is SOLID APPLE Key	
0	Modifier is OPEN-APPLE Key	

1. Updated by the DisableItem Command.
2. If the "Item is Filler" bit in the Option Byte is on, then Character 1 of the Menu Item Block (see Table H-8) is the character used for filler. If this is not the case, Character 1 and Character 2 are the upper and lower case values of the key which identifies the item when the MenuKey command is called.
3. Updated by the CheckItem command.
4. The program can set the initial states of these flags in the data structure before calling the SetMenu command. Afterwards, the program should only update the flags by calling the appropriate commands.

SetMenu (cont.)

Machine Language Commands:

```
SetMenu     equ  10         ; command number
sm.parms    db 1            ; parameter list for SetMenu
sm.mbar     dw  mymenu      ; pointer to Menu Data Structure
```

Pascal Interface:

```
Procedure SetMenu ( var my_menu_bar : menu_bar );
        my_menu_bar is the menu bar structure.
        MenuItem = Record
                        ItemOptB: byte;
                        MarkChar: byte;
                        Char1: byte;
                        Char2: byte;
                        ItemStrPtr: ^ItemStr
                   end;
        MenuData = Record
                        NumItems: byte;
                        reserve1: byte;
                        reserve2: byte;
                        reserve3: byte;
                        Items: array[ 1..MaxNumItems] of MenuItem
                   end;
        MenuTitle = Record
                        MenuID: byte;
                        Disabled: byte;
                        TitlePtr: ^TitleStr;
                        MDataPtr: ^MenuData;
                        reserve1: array[1..4] of byte
                    end;
        MenuBarPtr = ^MenuBar;
        MenuBar = Record
                        NumMenus: byte;
                        reserve1: byte;
                        Menus: array[1..MaxMenus] of MenuTitle
                  end;
```

ErrorCodes: 10 ($0A) Save area (from InitMenu) is too small.

SetUserHook

Function: SetUserHook sets the address of the user's interrupt handler.

Command Number: 47 ($2F)

Parameter List:

2	(input, byte)	the number of parameters
Id	(input, byte)	the ID number of the interrupt routine
Adr	(input, word)	the address of the interrupt routine

Description: The SetUserHook command sets the starting address of the application program's interrupt handler routine so that the Toolkit can pass control whenever the CheckEvent command is called. In Interrupt Mode, the Toolkit calls CheckEvent internally during interrupt servicing. Routines installed by the SetUserHook command become interrupt service routines for the application.

The CheckEvent command can pass control to the program's interrupt routine either before or after it checks events. The ID parameter determines at which point the CheckEvent command will call the interrupt routine. If ID=0, CheckEvent will call the interrupt routine before checking events, and if ID=1, CheckEvent will call the interrupt routine after checking events. In this way there can be an interrupt routine either before or after event checking.

The CheckEvent command will not check events if the interrupt routine which is called before event checking (ID = 0) returns to the Toolkit with the carry flag clear. This allows the application program to handle event checking itself and bypasses events checking by the Toolkit.

The SetUserHook removes any routine previously installed if the Adr parameter is set to 0.

Warning: The user interrupt routine can only call Toolkit commands PostEvent, ShowCursor, HideCursor, and SetCursor. Calling any other commands from the user interrupt routine could
put the Toolkit into an unknown and bizarre state.

SetUserHook (cont.)

Machine Language Commands:

```
SetUserHook   equ 47        ; command number
shook.parms   db 2          ; parameter
shook.id      db 0          ; user's routine ID
shook.addr    db 0          ; starting address of user's routine
```

Pascal Interface:

Procedure SetUserHook (hook_id, hook_adr: integer);

hook_id is the ID number (0 or 1) for program's interrupt routine.
hook_adr is the address of the program's interrupt routine.

Error Codes: 21 ($15) Illegal Id parameter (must be 0 or 1)

ShowCursor

Function: ShowCursor makes the cursor visible

Command Number: 3 ($03)

Parameter List:

0	(input, byte)	the number of parameters

Description: The ShowCursor command makes the cursor visible. If the cursor is temporarily invisible, ShowCursor has no effect.

Machine Language Commands:

```
ShowCursor    equ 3          ; command number
showc.parms   db 0           ; parameter list for ShowCursor
```

Pascal Interface: Procedure ShowCursor;

Error Codes: (none)

StartDeskTop

Function: StartDeskTop initializes the mouse and Toolkit routines.

Command Number: 0 ($00)

Parameter List:

6	(input, byte)	the number of parameters
id	(input, byte)	machine ID byte: $06 = Apple IIe or IIc
sid	(input, byte)	subsidiary ID byte: $EA = Apple IIe $E0 = Apple IIe with revised ROM $00 = Apple IIc
op	(input, byte)	operating system byte: 0 = ProDOS 1 = Pascal
s#	(input or output, byte)	slot number of the mouse card
int	(input or output, byte)	interrupt usage: 0 = Passive Mode only 1 = use interrupts
col	(input, byte)	number of text columns: 0 = 40 columns 1 = 80 columns

Description: The StartDeskTop command saves the current state of the computer, initializes the Toolkit routines, and activates the mouse card. If the calling program specifies slot number 0, StartDeskTop will check the slots for a mouse card and use the first slot it finds, returning its slot number in s#. If no mouse card is found, StartDeskTop will set the Passive Mode and return the int parameter as 0.

If the mouse card is required, the program should set the high bit of the s# parameter on before calling StartDeskTop. When the high bit is set, StartDeskTop will return an error condition if it doesn't find a mouse card.

If the program uses interrupts, it must set the int parameter to 1.

StartDeskTop (cont.)

The ID bytes are the values found at locations $FBB3 and $FBC0 in the Apple IIe and Apple IIc. The MouseText Toolkit requires the machine ID byte to be $06.

TheToolkit doesn't interact with the 80-column firmware. The application program must activate the firmware if it is needed.

StartDeskTop sets the cursor to the arrowhead character (ASCII value $02) and sets it hidden. After calling StartDeskTop, an application program can call ShowCursor immediately.

Machine Language Commands:

```
StartDesktop    equ 0          ; command number
start.parms     db 6           ; parameter list for StartDeskTop
start.mid       db 0           ; machine id byte
start.msid      db 0           ; machine subid byte
start.opsys     db $00         ; using ProDOS
start.slotn     db $00         ; slot no. for mouse (0 = check all slots)
start.int       db $01         ; using Interrupt Mode
start.col       db $01         ; using 80 columns
```

Pascal Interface:

```
Procedure StartDeskTop ( mach_id : integer;  sub_id: integer;
                         var slot_num : integer;  use_interrupts : boolean;
                         column_80 : boolean );
```

mach_id is the machine ID number.
sub_id is the subsidiary ID number.
slot_num is the slot number in the mouse card.
use_interrupts is the interrupt usage parameter:
    false= Passive Mode only
    true= use interrupts
column_80 is the col (number of text columns) parameter:
    false= 40 columns
    true= 80 columns

StartDeskTop (cont.)

Error Codes:

4 ($04)	Machine or operating system not supported
5 ($05)	Invalid slot # (less than 0 or greater than 7)
6 ($06)	Card not found
11 ($0B)	Could not install interrupt handler

StopDeskTop

Function: StopDeskTop deactivates the mouse and the Toolkit routines.

Command Number: 1 ($01)

Parameter List:

0	(input, byte)	the number of parameters

Description: The StopDeskTop command hides the cursor, removes the link to the interrupt handler, and sets the mouse to an inactive state. StopDeskTop then restores the computer to the initial state which was saved by StartDeskTop.

Important Note: In the MouseText Toolkit this procedure is incorporated into the include file called "StartDeskTop.I".

Machine Language Commands:

```
StopDesktop   equ 1          ; command number
stop.parms    db 0           ; parameter list for StopDeskTop
```

Pascal Interface: Procedure StopDeskTop;

Error Codes: (none)

TrackGoAway

Function: The TrackGoAway command tracks the mouse and indicates whether the mouse button was released in the Go-Away Box.

Command: 29 ($1D)

Parameter List:

1	(input, byte)	number of parameters
go	(output, byte)	Go-Away status 0 = not in Go-Away Box 1 = mouse in Go-Away Box

Description: The TrackGoAway command tracks the mouse until the mouse button is released. If the mouse is in the Go-Away Box when the button is released, the return status is 1. If the mouse is not in the Go-Away Box , the return status is 0.

The application program should call the TrackGoAway command when it detects the mouse button is down with the mouse in the Go-Away Box of the front window. If the return status indicates that the button was released in the Go-Away Box, the application program should then call the CloseWindow command.

Machine Language Commands:

```
TrackGoAway   equ  29        ; command number
tga.parms     db 1           ; parameter list for TrackGoAway
tga.closeit   db 0           ; Go-Away status
```

Pascal Interface

```
Procedure TrackGoAway ( var makeitgoaway: boolean );
        makeitgoaway is the go away status:
                0= not in the Go-Away Box
                1= mouse was in the Go-Away box
```

Error Codes: 16 ($10) There are no windows

TrackThumb

Function: The TrackThumb command tracks the thumb until the mouse button is released, then it updates the data in the Window Information Data Sructure.

Command Number: 41 ($29)

Parameter List:

3	(input, byte)	number of parameters
ctl	(input, byte)	control region whose Thumb is moving: 1 = vertical scroll bar 2 = horizontal scroll bar
pos	(output, byte)	position the Thumb moved to
stat	(output, byte)	return status: 0 = Thumb didn't move, pos is not valid 1 = Thumb did move

Description: The TrackThumb command tracks the Thumb until the mouse button is released. The application program should call the TrackThumb command when the FindControl command indicates that the mouse button is down in the Thumb. When the mouse button is released, the TrackThumb command updates the position information in the Winfo Data Structure and returns the new position of the Thumb. If the value of the return status is 0, the Thumb is in the same position it started in, and the value of pos is not valid.

The Thumb position is a number in a range from 0 to the maximum position on the horizontal or vertical scroll bar. A position of 0 means the first character of the document should be made visible. A position equal to the maximum value means the last character of the document should be made visible.

If the Thumb position is the same as it was when the TrackThumb command is called, it is treated as if it has not moved. If the Thumb does move, the TrackThumb command updates the Thumb position in the Winfo Data Structure.

The TrackThumb command operates only on the front window.

TrackThumb (cont.)

Machine Language Commands:

```
TrackThumb      equ  41         ; command number
tkthmb.parms    db 3            ; parameter list for TrackThumb
tkthmb.ctl      db 0            ; control region affected
tkthmb.pos      db 0            ; position Thumb moved to
tkthmb.moved    db 0            ; Thumb moved code
```

Pascal Interface

```
Procedure TrackThumb ( whichctl: ctlarea; var thumbpos: integer;
                       var thumbmoved: boolean );
    whichctl is the control region.
        CtlArea = (NotCtl, VerScroll, HorScroll, DeadZone)
    thumbpos is the Thumb position.
    thumbmoved is the return status:
        0 = Thumb didn't move, thumbpos not valid
        1 = Thumb did move
```

Error Codes:

16	($10)	There are no windows
18	($12)	Bad control ID (not 1 or 2)

UpDateThumb

Function: The UpDateThumb command redisplays the Thumb in the designated position.

Command Number: 42 ($2A)

Parameter List:

2	(input, byte)	number of parameters
ctl	(input, byte)	control region whose Thumb is being moved
pos	(input, byte)	new position of Thumb

Description: The UpDateThumb command redisplays the Thumb in the designated position and updates the position value in the Winfo Data Structure. The UpDateThumb command operates only on the front window.

The program should call the UpDateThumb command after scrolling or paging.

Machine Language Commands:

```
UpDate Thumb equ 42          ; command number

upt.parms    db 2            ; parameter list for UpdateThumb
upt.ctl      db 0            ; control region affected
upt.newpos   db 0            ; new position of Thumb
```

Pascal Interface

```
Procedure UpdateThumb (whichctl: ctlarea; thumbpos: integer );
        whichctl is the control region.
                CtlArea = (NotCtl, VerScroll, HorScroll, DeadZone)
        thumbpos is the new Thumb position.
```

Error Codes:

16 ($10) There are no windows
18 ($12) Bad control ID (not 1 or2)

Version

Function: Version returns the Toolkit's version and revision numbers.

Command Number: 19 ($13)

Parameter List:

2	(input, byte)	the number of parameters
Ver	(output, byte)	the version number of the Toolkit
Rev	(output, byte)	the revision number of the Toolkit

Description: The Version command returns the version and revision numbers of the Toolkit. The program can use these numbers to determine compatibility. The MouseText Toolkit Runtime Module is Version 2.0 released by Apple Computer Inc. The Version routine will be used only if you write application programs which might be affected by a future revision of the Runtime Module by Apple.

Machine Language Commands:

```
Version     equ 19      ; command number
ver.parms   db 2        ; parameter list for Version
ver.ver     db 0        ; version number
ver.rev     db 0        ; revision number
```

Pascal Interface:

```
Procedure Version ( var ver_num, rev_num: integer);
        ver_num is the version number.
        rev_num is the revision number.
```

Error Codes: (none)

WinBlock

Function: The Win Block command writes a block of text in a window.

Command Number: 36 ($24)

Parameter List:

6	(input, byte)	number of parameters
id	(input, byte)	ID number of window
ptr	(input, word)	pointer to Document Information Data Structure for the text to be displayed. If ptr = 0, WinBlock uses the Dinfo pointer from the Winfo specified by the window ID (See OpenWindow).
startx	(input, word)	X coordinate of upper-left corner of display window position within the document window.
starty	(input, word)	Y coordinate of upper-left corner of display window position within the document window.
stopx	(input, word)	X coordinate of lower-right corner of display window position within the document window.
stopy	(input, word)	Y coordinate of lower-right corner of display window position within the document window.

Description: The WinBlock command writes a block of text in a window. Startx, starty, stopx, and stopy define a rectangle in the window where characters are displayed. The WinBlock command does not alter any information outside this rectangle.

The WinBlock command does not update the document.

Setting ID = 0 selects the front window.

WinBlock (cont.)

Machine Language Commands:

```
WinBlock      equ 36        ; command number

wblk.parms    db 6          ; parameter list for WinBlock
wblk.id       db 0          ; window ID number
wblk.ptr      dw 0          ; pointer to Dinfo Data Structure
wblk.x1       dw 0          ; X upper-left window coordinate
wblk.y1       dw 0          ; Y upper-left window coordinate
wblk.x2       dw 0          ; X lower-right window coordinate
wblk.y2       dw 0          ; Y lower-right window coordinate
```

Pascal Interface

```
Procedure WinBlock ( window_id: integer; var my_dinfo: dinfo;
                     startx, starty, stopx, stopy: integer );
```

window_id is the window ID number.
my_dinfo is the document information structure.
startx is the X coordinate of the upper-left corner.
starty is the Y coordinate of the upper-left corner.
stopx is the X coordinate of the lower-right corner.
stopy is the Y coordinate of the lower-right corner.

```
Dinfo = Record
        DocPtr: integer;
        Reserve1: byte;
        DocWidth: byte;
        DocX: integer;
        DocY: integer;
        DocLength: integer;
        Reserve2: integers;
        Reserve3: integer
       end;
```

Error Codes: 15 ($0F) Window ID not found

WinChar

Function: The WinChar command writes a character in a window.

Command Number: 34 ($22)

Parameter List:

4	(input, byte)	number of parameters
id	(input, byte)	ID number of window
wx	(input, word)	X coordinate in window
wy	(input, word)	Y coordinate in window
char	(input, byte)	character to display

Description: The WinChar command writes a character in a window at a given position. If the position given is not inside the window, the WinChar command does not write the character.

The WinChar command does not update the document. Setting ID = 0 selects the front window.

Machine Language Commands:

```
WinChar     equ 34      ; command number
wch.parms   db 4        ; parameter list for WinChar
wch.id      db 0        ; window ID number
wch.wx      dw 0        ; X coordinate in window
wch.wy      dw 0        ; Y coordinate in window
wch.char    db $00      ; ASCII character to display
```

Pascal Interface

```
Procedure WinChar ( window_id, windowx, windowy: integer;
                    my_char: char );
```

window_id is the window ID number.
windowx is the window X coordinate.
windowy is the window y coordinate.
my_char is the character to display.

Error Codes: 15 ($0F) Window ID not found

WindowScreen

Function: The WindowScreen command converts window coordinate values to screen coordinates.

Command Number: 32 ($20)

Parameter List:

5	(input, byte)	number of parameters
id	(input, byte)	ID number of window to use
wx	(input, word)	X coordinate in the window
wy	(input, word)	Y coordinate in the window
sx	(output, word)	X coordinate for the screen
sy	(output, word)	Y coordinate for the screen

Description: The WindowScreen command converts passed coordinate values from window coordinates to screen coordinates. Setting ID = 0 selects the front window.

Machine Language Commands:

```
WindowScreen equ  32        ; command number
w2s.parms    db 5           ; parameter list
w2s.id       db 0           ; window ID number
w2s.wx       dw 0           ; X coordinate in window
w2s.wy       dw 0           ; Y coordinate in window
w2s.sx       dw 0           ; X screen coordinate
w2s.sy       dw 0           ; Y screen coordinate
```

Pascal Interface

```
Procedure WindowScreen (window_id, windowx, windowy: integer;
                        var screenx, screeny: integer );
```

window_id is the window ID number.
windowx is the window X coordinate.
windowy is the window Y coordinate.
screenx is the screen X coordinate.
screeny is the screen Y coordinate.

Error Codes: 15 ($0F) Window not found

WinOp

Function: The WinOp command performs an operation on a window.

Command Number: 37 ($25)

Parameter List:

4	(input, byte)	number of parameters
id	(input, byte)	ID number of window
wx	(input, word)	X window coordinate
wy	(input, word)	Y window coordinate
op	(input, byte)	operation to perform:
		26 ($1A) = clear to start of window*
		27 ($1B) = clear to start of line*
		28 ($1C) = clear window
		29 ($1D) = clear to end of window
		30 ($1E) = clear line
		31 ($1F) = clear to end of line

* Operations do not clear the character at position X,Y.

Description: The WinOp command clears all or a portion of a window, depending on the specific operation code. Except for operation code 28 (ClearWindow), the WinOp command clears the characters from position X,Y to the end of the area indicated by the operation. The forward "clears" include the character at position X,Y , but the backward "clears" (i.e., clear to start of the window and clear to start of the line)do not. These operations clear from the start of an area, up to but not including, position X,Y.

Setting ID = 0 selects the front window.

Machine Language Commands:

```
WinOp       equ 37      ; command number

wop.parms   db  4       ; parameter list for WinBlock
wop.id      db  0       ; window ID number
wop.wx      dw  0       ; X window coordinate
wop.wy      dw  0       ; Y window coordinate
wop.op      db  0       ; window operation
```

WinOp (cont.)

Pascal Interface

```
Procedure WinOp  ( window_id, windowx, windowy: integer;
                   opcode: byte);
```

window_id is the Window ID number.
windowx is the window X coordinate.
windowy is the window Y coordinate.
opcode is the code for the operation to perform.

Error Codes: 15 ($0F) Window ID not found

WinString

Function: The WinString command writes a string in a window.

Command Number: 35 ($23)

Parameter List:

5	(input, byte)	number of parameters
id	(input, byte)	ID number of window
wx	(input, word)	X coordinate in window
wy	(input, word)	Y coordinate in window
ptr	(input, word)	pointer to the string
res	(input, byte)	must be 0.

Description: The WinString command writes a string in a window at a given position. The WinString command does not wrap around. If the string extends past the right edge of the window, the WinString command truncates it. The WinString command does not display any characters in the string that fall outside the edges of the window.

The WinString command does not update the document.

Setting ID = 0 selects the front window.

Machine Language Commands:

```
WinString       equ  35          ; command number

wstr.parms      db  5            ; parameter list for WinString
wstr.id         db  0            ; window ID number
wstr.wx         dw  0            ; X coordinate in window
wstr.wy         dw  0            ; Y coordinate in window
wstr.ptr        dw  0            ; pointer to string
wstr.res        db  0            ; reserved
```

WinString (cont.)

Pascal Interface:

Procedure WinString (window_id, windowx, windowy: integer;
my_string: string);
window_id is the window ID number.
windowx is the window X coordinate.
windowy is the window Y coordinate.
my_string is the string to write.

Error Codes: 15 ($0F) Window ID not found

WinText

Function: The WinText command writes ASCII characters in a window.

Command Number: 38 ($26)

Parameter List:

5	(input, byte)	number of parameters
id	(input, byte)	ID number of window
wx	(input, word)	X coordinate in window
wy	(input, word)	Y coordinate in window
ptr	(input, word)	pointer to the first character of text
len	(input, byte)	number of characters to display

Description: The WinText command writes ASCII characters at a given position in a window. The WinText command does not wrap around. If the characters extend past the right edge of the window, the WinText command truncates them. The WinText command does not display any characters that fall outside the edges of the window.

The WinText command does not update the document.

Setting ID = O selects the front window.

Machine Language Commands:

```
WinText      equ  38      ; command number

wtxt.parms   db  5        ; parameter list for WinString
wtxt.id      db  0        ; window ID number
wtxt.wx      dw  0        ; X coordinate in window
wtxt.wy      dw  0        ; Y coordinate in window
wtxt.ptr     dw  0        ; pointer to first character
wtxt.len     db  0        ; number of characters
```

WinText (cont.)

Pascal Interface:

```
Procedure WinText ( window_id, windowx, windowy, text_buffer,
                    textlength: integer );
```

window_id is the window ID number.
windowx is the x coordinate in the window.
windowy is the y coordinate in the window.
text_buffer is the pointer to the first character of text.
textlength is the number of characters to display.

Error Codes: 15 ($0F) Window ID not found

I. Pascal Data Structures

This chapter presents the specifications of the data types and data structures used in the MouseText Toolkit, including the Menu Data Structure, the Window Information Data Structure, and the Document Information Data Structure.

The following constants and data types are used in the Pascal Interface.

Constants

MaxMenus = 3 (A maximum of 3 menus is supported).
MaxTitleStr = 12 (A maximum of 12 characters per menu title)
MaxItemStr = 12 (A maximum of 12 characters per menu item name)
MaxNumItems = 7 (A maximum of 7 menu items is supported).
SaveSize =256.

The following event type values are provided as constants rather than as an enumerated type so that the user can define and handle their own events.

```
No_Event = 0
Button_Down = 1
Button_Up = 2
Key_Down = 3
Drag = 4
Apple_Key = 5
Update_Event = 6
```

A single byte value is defined as: byte = char;

Event

An event is defined as:

```
type_event = record
             evtkind : byte;
             char1 : byte;
             char2 : byte;
             reserve1 : byte
             end;
```

where:

evtkind is the event type value (see above under Constants).
char1 is event byte 1, X coordinate or key value.
char2 is event byte 2, Y coordinate or key modifier.
reserve1 is reserved for use by the ToolKit.

Menu titles are defined as:

```
TitleStr = array[1..MaxTitleStr] of char;
```

Menu Item Names

Menu item names are defined as:

```
ItemStr = array[1..MaxItemStr] of char;
```

Menu Item Blocks

A Menu item block is defined as:

```
MenuItem = record
        ItemOptB : byte;
        MarkChar : byte;
        char1 : byte;
        char2 : byte;
        ItemStrPtr : ^ItemStr
        end;
```

where:

ItemOptB (the first 8 bits):
open_apple is on when modifier is OPEN-APPLE key; {bit 0}
solid_apple is on when the modifier is SOLID-APPLE key;
item_has_mark is on when the item has mark;
reserve2, reserve3 are reserved for use by the ToolKit;
item_is_checked is on when the Item Is Checked;
item_is_filler is on when the Item Is Filler;
disable_flag is the Disable Flag; {bit 7}

markchar is the mark character;

char1 is Character 1;
char2 is Character 2;

ItemStrPtr is Pointer to Item String;

Menu Data Structures

The Data Structure for a menu is defined as:

```
MenuData = record
        NumItems : byte;
        reserve1 : byte;
        reserve2 : byte;
        reserve3 : byte;
        items : array [1..MaxNumItems] of MenuItem
        end;
```

where:

NumItems is the Number of Items;
reserve1, reserve2, reserve3 are reserved for ToolKituse;
items is the array of Menu Item Blocks;

Menu Title Blocks

A Menu Title Block is defined as:

```
MenuTitle = record
        MenuId : byte;
        Disabled : byte;
        TitlePtr : ^TitleStr;
        MDataPtr : ^MenuData;
        reserve1 : array [1..4] of byte
        end;
```

where:

menuid is the Menu ID;

disabled is the Disable Flag (only bit 7 can be used);
titleptr is the Pointer to Title String;
mdataptr is the Pointer to Menu Data Structure;
reserved is reserved by the ToolKit;

Menu Bars

The menu bar is defined as:

```
MenuBar = record
        NumMenus : byte;
        reserve1 : byte;
        Menus : array [1..MaxMenus] of MenuTitle
        end;
```

where:

nummenus is the Number of Menus;
reserved is reserved for use by the ToolKit;
menus is the array of Menu Blocks;

MenuBarPtr = ^MenuBar;

WIndow Information Data Structures

A Window Information Data Structure is defined as:

```
Winfo = record
        WindowId : byte;
        WinOpt : byte;
        TitlePtr : ^TitleStr;
        WindowX : integer;
        WindowY : integer;
        ContWidth : byte;
        ContLength : byte;

        MinContWidth: byte;
        MaxContWidth: byte;
        MinContLength: byte;
        MaxContLength: byte;
```

```
        DinfoPtr: ^Dinfo;
        HorContOpt : byte;
        VertContOpt : byte;

        HThumbMax : byte;
        HThumbPos : byte;
        VThumbMax : byte
        VThumbPos : byte
        WinStatus : byte;
        Reserve1 : byte;
        NextWinfo = ^Winfo

        Reserve2 : byte;
        Reserve3 : byte;
        Reserve4 : byte;
        Reserve5 : byte;
        Reserve6 : byte;
        Reserve7 : byte;
        end;
```

where:

windowid is the Window ID #

WinOpt

bit 0 is dialog/alert window flag
bit 1 is on when Go-Away Box present
bit 2 is on when Grow Box present
bit 7 is user routine adr/dinfo ptr

titleptr is Title Str ptr

windowx is WIndow Location X
windowy is Window Location Y

contwidth is Current Content Width
contlength is Current Content Length

mincontwidth is Min Content Width
maxcontwidth is Max Content Width
mincontlength is Min Content Length
maxcontlength is Max Content Length

dinfoptr is Dinfo Ptr

HorContOpt
bit 0 is on when horizontal scrollbar active
bit 6 is on when horizontal Thumb present
bit 7 is on when horizontal scroll bar present
VertContOpt
bit 0 is on when vertical scroll bar active
bit 6 is on when vertical Thumb present
bit 7 is on when vertical scroll bar present

hthumbmax is horizontal scroll maximum
hthumbpos is current horizontal Thumb position
vthumb is vertical scroll maximum
vthumb is current is current vertical Thumb position

WinStatus
bit 7 is window open

nextwinfo is the pointer to the next winfo structure

Winfo_Ptr = ^Winfo;

Document Information Data Structures

A Document Information Data Structure (Dinfo) is defined as:

```
dinfo = record
        DocPtr : integer
        Reserve1 : byte;
        DocWidth : byte;
        DocX : integer;
        DocY : integer;
        DocLength: integer;
        Reserve2: byte;
        Reserve3: byte
        end;
```

where:

docptr is Document ptr
reserved is reserved by the ToolKit
docwidth is Document Width
docx is Document X
docy is Document Y
doclength is Document Length
reserve2, reserve3 are reserved by the ToolKit

```
Dinfo_ptr = ^Dinfo;
```

Screen Region Types

The type of screen region is defined as:

```
Type_Area = (InDeskTop, InMenubar, InContent,
        InDrag, InGrow, InGoAway);
```

where each value is as returned by FindWindow:

inDeskTop is in desktop
inMenubar is in menu bar
inContent is in content region
inDrag is in drag region
inGrow is in Grow Box
inGoAway is in Go-Away Box

Control Region Types

The type of control region is defined as:

```
CtlArea = ( NotCtl, VerScroll, HorScroll, DeadZone );
```

where each value is as returned by FindControl:

notctl is in content region
verscroll is in vertical scroll bar
horscroll is in horizontal scroll bar
deadzone is none of the above

Control Region Part Types

The type of a part of a control region is defined as

```
CtlPart = ( CtlInactive, ScrollUpLeft, ScrollDownRight,
        PageUpLeft, PageDownRight, Thumb );
```

where each value is as returned by FindControl:

ctlinactive is never returned
scrollupleft is up arrow of vertical scroll bar
or Left-Arrow of horizontal scroll bar
scrolldownright is Down-Arrow of vertical scroll bar
or Right-Arrow of horizontal scroll bar
pageupleft is "page up" region of vertical scroll bar
or "page left" region of horizontal scroll bar
pagedownright is "page down" region of vertical scroll bar
or "page right" region of horizontal scroll bar
thumb is Thumb of scroll bar

Appendix I
The AppleMouse II Interface Card

To use the AppleMouse with an Apple //, Apple //+, or Apple //e, you need the AppleMouse II Interface Card installed in one of the expansion slots (slot 4 is recommended). Like most Apple peripheral cards, it contains I/O firmware that is executed by the 6502 central processor whenever you access the slot. The mouse interface card also contains its own microprocessor with firmware and a timer. The microprocessor on the card keeps track of the position of the mouse and the state of the button on the mouse. The microprocessor handles the transfer of mouse information and other communications between the card and the central processor.

Passive Versus Active Operation

Most positioning devices used with the Apple II, such as the joystick and the graphics tablet, are passive devices; that is, they don't require any processing until an application program requests information from them. The mouse, on the other hand, is an active device; movement of the mouse requires near-constant attention to keep the system from losing track of its position and direction.

A computer normally uses interrupts to handle this need for immediate response. When the mouse is moved rapidly, it generates interrupts often enough to have a significant impact on the computer's operation. If the computer is engaged in other tasks that are dependent on precise timing, as the Apple II often is, the added burden of processing the interrupts from the mouse can be intolerable.

To reduce the interrupt burden on the Apple II's processor, the AppleMouse II uses an intelligent interface card. The card has an MC6805 microprocessor that is dedicated to keeping track of the mouse, thus making it possible for the AppleMouse II to operate as either an active device or a passive device. In the Passive Mode, the MC6805 determines the instantaneous movement and direction of the mouse and stores the information on the card until the processor in the Apple II requests the information. Thus, the AppleMouse II can act like a passive device in applications that cannot tolerate interrupts, or, for

applications where interrupts are appropriate, it can operate as an active device.

Mouse Interrupts

One reason to use the mouse in Interrupt Mode is to be able to move a cursor on the display screen without the flicker produced by updating the cursor during the wrong part of the display refresh cycle. In Interrupt Mode, the Applemouse II generates interrupts that are synchronized with the vertical blanking interval.

The Apple IIe has a signal named VBL, but it isn't available as an interrupt. The VBL signal is not available at all on an Apple II or Apple II Plus, so the mouse card has a hardware timer that it uses to generate interrupts synchronized with the vertical blanking interval.

Because the AppleMouse II transmits an interrupt request only at the beginning of a vertical blanking interval, it cannot generate interrupts faster than 60 times per second. This limits the number of mouse interrupts and keeps the mouse from monopolizing the central processor.

The TimeData Firmware Call

There is a little-used call in the firmware on the AppleMouse II card. That call sets the interrupt rate to either 50 or 60 Hz. The default is 60 Hz.,which keeps the VBL interrupts the card generates in step with the true VBL on a North American Apple II. For European machines, the VBL rate is 50 Hz.

The low byte of the TimeData entry-point address is $Cn1C. Input data is in the accumulator. With the accumulator set to 0, TimeData sets the VBL rate to 60 Hz. With the accumulator set to 1, the call sets the VBL rate to 50 Hz. The only valid accumulator contents for this call are 0 and 1. On out, the carry bit is clear and the screen holes are unchanged.

You should call TimeData just before calling InitMouse. If you do not call TimeData first, the VBL rate will be set to 60 Hz. when you call InitMouse.

Appendix II
The Mouse Firmware Interface

On the Apple IIc, the interface hardware and firmware for the AppleMouse II is built in. On the Apple IIe, the user must install a mouse interface card in order to use the AppleMouse II. The interface card for the AppleMouse II contains the firmware that communicates with and controls the mouse hardware.

The Apple II MouseText Toolkit uses the mouse firmware in the Apple IIc or in the card in the Apple IIe to operate the mouse. This appendix describes the interface to the firmware.

Note: If you do all your mouse operations via Toolkit commands, you do not need to communicate directly with the mouse firmware and so do not need to learn the material in this appendix..

Finding the Mouse Card

The AppleMouse II interface card can be installed in any peripheral slot except slot 0; use of slot 4 is recommended but not required. The firmware on the card stores signature bytes in five of the memory locations assigned to the slot it is in. The addresses and values of the signature bytes are as follows:

Address	Value
$Cn05	$38
$Cn07	$18
$Cn0B	$01
$Cn0C	$20
$CnFB	$D6

The letter n in the addresses stands for the slot number. Your program can determine which slot the mouse card is in by reading the memory locations for each value of n from 1 to 7 and comparing the values with the values shown above.

Reading Mouse Data

The mouse firmware stores position and status information in the display buffer locations reserved for the slot the mouse card is in (the screen holes, also called mouse holes). When you call the Readmouse routine or the ServeMouse routine (described later in this appendix), the firmware updates the information in the mouse holes. Your program can address these locations by using the slot number as an index, as indicated by the letter n in Table J-1.

Note: Chapter 6 of the Apple IIe Reference Manual describes the way you address the reserved screen locations.

Warning: If your program ever uses the auxiliary memory in the Apple IIe, be sure that you get all the switches set back to main memory before you use the Toolkit. If you write data into the reserved screen locations in the auxiliary memory, not only will the mouse firmware not read them,but you may cause other firmware to malfunction (spelled c-r-a-s-h).

Table J-1. Screen Locations for Mouse Data

Address	Contents
$478 + n	Low byte of X position
$4F8 + n	Low byte of Y position
$578 + n	High byte of X position
$5F8 + n	High byte of Y position
$678 + n	(used by the firmware)
$6F8 + n	(used by the firmware)
$778 + n	Button and interrupt status
$7F8 + n	Current Operating Mode

In its normal operating position (oriented with its cable directed away from the user), the value of the X position coordinate increases as a mouse is moved to the right and the value of the Y position coordinate increases as the mouse is moved toward the user. The maximum values of X and Y are -32768 to +32767, but the firmware normally clamps them to a range 0 to +1023 ($0 to $3FF). You can change the

clamping range by calling the ClampMouse routine, which is described later in this appendix.

The smallest mouse movement that the mouse hardware can detect is one count in either the X or Y direction; that is equivalent to about 0.01 inch (0.3 mm). The largest movement that the hardware can handle is 16 bits in either axis. A change of position from -32768 to 32767 corresponds to about 60 feet of mouse movement.

The bits in the button and interrupt status byte are assigned as shown in Table J-2, where a value of 1 means the function is true.

Table J-2. Button and Interrupt Status Byte

Bit #	Function
7	Button is down
6	Button was down
5	Mouse moved since last reading
4	(used by the firmware)
3	Video blanking interrupt
2	Button press interrupt
1	Mouse movement interrupt
0	(used by the firmware)

Operating Modes

When you turn on the power, the firmware comes up in the off condition with its X and Y position register set to 0. you activate the firmware by loading the accumulator with a mode byte and calling the SetMouse routine. The settings of the bits in the mode byte determine the mode of operation, as shown in Table J-3.

Table J-3. Bits in the Mode Byte

Bit #	Function
7-4	(used by the firmware)
3	Enable interrupt on video blanking (VBL)
2	Enable interrupt on next VBL after button pressed
1	Enable interrupt on next VBL after mouse movement
0	Turn on the mouse

You can enable any combination of interrupts by setting the appropriate bits in the mode byte. you can set mode combinations that don't make sense, such as $02: Mouse Off plus Enable Interrupt On Mouse Movement, which acts just like $00: Mouse Off.

Setting the low bit in the mode byte to 0 turns off certain functions of the mouse: the mouse position is not tracked, calls to Readmouse don't update the status byte or the screen holes, and button and movement interrupts are not generated. Other mouse functions will work as usual: PosMouse and ClearMouse will change the mouse position data, ClampMouse will set new values, and so on. Turning the mouse on and off by changing the mode byte does not reset any mouse values.

Warning: You must not set the high bits of the mode byte. Mode byte values greater than $0F will cause the SetMode routine to return an illegal-mode error.

Passive Mode

Calling the SetMouse routine with a mode byte of $01 puts the firmware into Passive Mode (no interrupts occur). Passive mode is the simplest way to use the mouse, and it is the only way to use it in systems with peripherals that cannot tolerate interrupts.

In Passive Mode, the interface card stores mouse information without affecting the operation of the CPU. When your program calls the Readmouse routine, the firmware updates the mouse information in the screen locations, where your program can read it.

Interrupt Mode

If your program uses interrupt, it must include an interrupt handling routine that calls the ServeMouse routine. The ServeMouse routine determines whether the interrupt was caused by the mouse. If it was, the ServeMouse routine calls ReadMouse.

Depending on the setting of the mode byte, the firmware can interrupt the CPU on one or more of the following events:

- o Mouse motion
- o Mouse button pressed
- o Display video blanking

You can set the mode byte to $08 -- mouse off, VBL interrupt on -- to generate interrupts on display video blanking (VBL) only. Regardless of the kind of event that causes the interrupt, the mouse hardware will interrupt the CPU only at the beginning of the video blanking interval, which occurs every 60th of a second. This enables your program to update the display between screen refresh cycles and avoid making the display flicker.

Unclaimed Interrupts

There is a bug in the AppleMouse II firmware that can effect the way ServeMouse Works. If the application program takes more than one video blanking cycle (normally about 16 milliseconds) to respond to a mouse-generated interrupt, there is a chance that ServeMouse will not claim the interrupt. In a ProDOS environment, this can be fatal. There are several possible ways to avoid this problem.

One approach, if you are not working under a system like ProDOS, is to make sure that unclaimed interrupts aren't fatal to your system and just ignore them. Another solution is to make sure that you always service interrupts within one VBL cycle (one sixtieth of a second). If you have to turn off interrupts for that long or longer, you should first use SetMouse to set the mode to 0 and call ServeMouse to clear any existing interrupt.

If you are working under an established operating system, like ProDOS, for which unclaimed interrupts are fatal, you can use one of the following suggestions to make sure that all interrupts are claimed.

If the mouse is the only interrupting device, write your interrupt handler so that it claims all interrupts.

If the mouse is not the only interrupting device, there are three ways of handling the problem. One is to write the mouse interrupt handler to claim all unclaimed interrupts and make sure that it is installed last. Another method is to write a spurious interrupt handler (sometimes called a demon), not associated with any device, that claims all unclaimed interrupts. This interrupt handler must be installed last. The third method is to include code in every interrupt handler to determine whether that interrupt handler is last. If it is, then that interrupt handler claims any unclaimed interrupts, even if not generated by its device.

Making Calls to Mouse Firmware

Your programs make calls to the mouse firmware by means of a table that conforms to Apple Firmware Protocol 1.1, described in the Apple IIe Design Guidelines as Pascal 1.1 Protocol. Table J-4 contains the low byte of the entry address of each of the firmware routines. (The high byte of each address is $Cn, where n is the number of the slot the mouse interface card is in.) The address bytes are stored in locations $Cn12 through $Cn19, arranged as shown in Table J-4.

Table J-4. Entry Point Address Bytes

Location	Contents
$Cn12	Low byte of SetMouse entry-point address
$Cn13	Low byte of ServeMouse entry-point address
$Cn14	Low byte of ReadMouse entry-point address
$Cn15	Low byte of ClearMouse entry-point address
$Cn16	Low byte of PosMouse entry-point address
$Cn17	Low byte of ClampMouse entry-point address
$Cn18	Low Byte of HomeMouse entry-point address
$Cn19	Low byte of InitMouse entry-point address

Thus, for a mouse card installed in slot 4, you can calculate the entry address for the SetMouse routine by adding $C400 to the contents of location $C412. Your program can use the values in the table to construct a jump table to use for calling the routines.

By the way: You must disable interrupts before calling the mouse firmware.

Parameter Passing

Before calling any of the firmware routines, your program must load the X and Y index registers with the number of the slot the mouse card is in, as follows:

X index register: $Cn
Y index register: $n0

Your program passes information to certain firmware routines via the accumulator and the screen locations, as noted in the descriptions of the routines.

When your program regains control, the contents of the accumulator and the index registers will be undefined, except as noted in the descriptions of the routines. The carry bit indicates the error status of the routine just ended:

Successful execution: C = 0
Unsuccessful execution: C = 1

The Firmware Routines

This section describes the functions of the firmware routines whose entry-point addresses are given in the previous section.

SetMouse

SetMouse starts the mouse operating in the mode indicated by the contents of the accumulator, as defined in the "Operating Modes" section earlier in this appendix. If the mode byte is greater than $0F, the routine will return with the carry bit set to one, indicating an error.

This routine does not clear the screen locations used for storing mouse data.

ServeMouse

If the pending interrupt was caused by the mouse, ServeMouse sets the status byte at location $778 + n to show what event caused the interrupt. Upon return from this routine, the carry bit is set to 0 if the interrupt was caused by the mouse; otherwise, the carry bit is set to 1. This routine does not update the other mouse screen locations.

Note: This routine is an interrupt service routine; it does not require particular values in the accumulator or the index register.

ReadMouse

Readmouse transfers the current values of the mouse X and Y position and button data into the appropriate screen locations and sets bits 1, 2, and 3 of the status byte at location $778 + n to 0. On return, the carry bit is 0.

ClearMouse

ClearMouse sets the mouse's X and Y position values to zero, both on the interface Card and in the screen locations. It does not change the contents of the interrupt and button status byte. On return, the carry bit is 0.

PosMouse

PosMouse sets the mouse X and Y position to the values in the screen locations. On return, the carry bit is 0.

Warning: Do not change the contents of any screen locations other than the X and Y position locations.

ClampMouse

ClampMouse sets the clamping bounds for either the X or Y position value. To clamp the X direction, load the accumulator with a); to clamp the Y direction, load the accumulator with a 1. Store the new bounds in the slot 0 screen locations, as follows:

$478	low byte of lower clamping bound
$4F8	low byte of upper clamping bound
$578	high byte of lower clamping bound
$5F8	high byte of upper clamping bound

On return, the carry bit is 0 and the X and Y position screen locations are undefined. To get valid position data, you have to call the ReadMouse routine.

HomeMouse

HomeMouse sets the internal position values to the upper-left corner of the clamping window. On return, the carry but is 0 and the X and Y screen locations are changed.

InitMouse

InitMouse sets internal mouse data to default values and synchronizes the interrupt timer on the card with the display vertical blanking. On return, the carry bit is zero and the screen locations are unchanged. To get valid position data, you have to call the ReadMouse routine.

Warning: On the Apple II plus, the InitMouse routine clears the Hi-Res screen in order to synchronize its timer with the vertical blanking, so you should display Hi-Res graphics only after you have called InitMouse.

Appendix III
ToolKit Error Codes

Table J-5 is a cumulative list of the error codes returned in the 6502's accumulator when a MouseText Toolkit command encounters an error condition. The error codes returned by each command are listed with the commands in Section H.

In addition to the error codes returned by individual commands, the first three listed here are generic error codes that can be returned by any command.

Table J-5. Mousetext Toolkit Error Codes

1 ($01)	Illegal Command number
2 ($02)	Wrong number of parameters
3 ($03)	StartDeskTop hasn't been called
4 ($04)	Machine or operating system not supported
5 ($05)	Invalid slot number (less than 0 or greater than 7)
6 ($06)	Mouse Interface Card not found
7 ($07)	Interrupt mode in use (Program specified interrupt mode in StartDeskTop, so it can't call CheckEvents.)
8 ($08)	Menu ID was not found
9 ($09)	Item Number is not valid
10 ($0A)	Save area (from InitMenu) is too small
11 ($0B)	Toolkit could not install interrupt handler
12 ($0C)	Window with same ID already open
13 ($0D)	InitWindowMgr buffer too small for this window
14 ($0W)	Bad Winfo -- tried to open window with ID = 0, or conflicting max and min width or length
15 ($0F)	Window ID number not found
16 ($10)	There are no windows
17 ($11)	Error returned by user hook routine
18 ($12)	Bad control ID (not 1 or 2)
19 ($13)	Event queue full, event not posted
20 ($14)	Illegal event, event not posted
21 ($15)	Illegal UserHook ID number (not 0 or 1)
22 ($16)	Operation cannot be performed

Appendix IV
Toolkit Disk Organization

The MouseText Toolkit is shipped on a "flippy" disk. The volume name of each side is /MouseText/. The contents of side one are in the root directory. The contents of side two are contained in the directory "PascalTools". The contents of each side are listed below.

/MOUSETEXT/ **Side 1**

FILENAME	FILETYPE	DESCRIPTION
Demo2	Bin	Demo program (Object code)
Demo2.S	Text	Demo program (Source code)
Exit.I	Text	KIX exit routine
KIX.SYSTEM	System	KIX System File
Load.I	Text	Utility program
Makeabs	$FC	Utility program
Mtxkit.abs	Bin	MouseText Runtime Module
Mtxkit.obj	$FE	MouseText Runtime Module
ProDOS	System	ProDOS Operating System
Rel2abs	Bin	Utility program
Tokniz.I	Text	Utility program

/MOUSETEXT/ **Side 2**

FILENAME	FILETYPE	DESCRIPTION
PascalTools	**Dir**	**Disk Directory**
Demo1	Bin	Demo program (object code)
Demo1.P	Text	Demo program (source code)
Demo1a	Bin	Demo program (object code)
Demo1a.P	Text	Demo program (source code)
DemoWindow.I	Text	Demo include file
LIB	Bin	Pascal Runtime Library
Mtxkit.abs	Bin	MouseText Runtime Module
Mtxkit.con	Text	MouseText Constants
Mtxkit.typ	Text	MouseText Types
P.out	Text	Intermediate Assembly file
Bload.I	Text	Pascal utility program
GetMachID	Text	Utility program
KeyBoardMouse.I	Text	Startup Commands
SetUserHook.I		
PascIntAdr.I		
StartDeskTop.I		
Version.I		
HideCursor.I	Text	Cursor Commands
ObscureCursor.I		
SetCursor.I		
ShowCursor.I		
CheckEvent.I	Text	Event-Handling Commands
FlushEvent.I		
GetEvent.I		
PeekEvent.I		
PostEvent.I		
SetKeyEvent.I		

/MOUSETEXT/ **Side 2 (cont.)**

FILENAME	FILETYPE	DESCRIPTION
CheckItem.I	Text	Menu Commands
DisableItem.I		
DisableMenu.I		
HiLiteMenu.I		
InitMenu.I		
MenuKey.I		
MenuSelect.I		
SetMark.I		
SetMenu.I		
CloseAll.I	Text	Window Commands
CloseWindow.I		
DragWindow.I		
FindWindow.I		
FrontWindow.I		
GrowWindow.I		
GetWinPtr.I		
InitWindowMgr.I		
OpenWindow.I		
ScreenWindow.I		
SelectWindow.I		
TrackGoAway.I		
WinBlock.I		
WinChar.I		
WinOp.I		
WindowScreen.I		
WinString.I		
WinText.I		
ActivateCtl.I	Text	Control Commands
FindControl.I		
SetCtlMax.I		
TrackThumb.I		
UpdateThumb.I		

Appendix V
Other Pointing Devices

The Toolkit supports other pointing devices (besides mice). Any hardware that appears to the computer as a mouse will be treated like a mouse and will work with the Toolkit. Other hardware can be supported by attaching a driver to the Toolkit via the AttachDriver call.

When a driver is attached, the Toolkit calls the driver whenever it would ordinarily call the mouse. The Toolkit calls the mouse at the following times.

- o The mouse is inited and turned on when desktop is started.
- o The mouse is homed when the desktop is started.
- o The mouse is set (turned off) when the desktop is stopped.
- o The mouse is served (in interrupt mode) on each interrupt (whether or not it was generated by the mouse).
- o Each time CheckEvents is called, the mouse is read.
- o During initialization the mouse is clamped.
- o When the scale factors are changed, the mouse is clamped.
- o When a mouse event is posted, the mouse is set.
- o During Keyboard mouse activities (both safety net and others), the mouse is set.

The device driver must be able to support the followng calls:

SetMouse	$12
ServeMouse	$13
ReadMouse	$14
ClearMouse	$15
PosMouse	$16
ClampMouse	$17
HomeMouse	$18
InitMouse	$19

The driver is called at the address passed in the AttachDriver call with the call number listed above in the Y register. The A register also contains information for ClampMouse and SetMouse. For ClampMouse the A register holds zero for clamping in the X direction and holds 1 for clamping in the Y direction. For SetMouse, the A register holds the mouse modes as follows:

Table J-6. A Register Contents

Bit #	Function
7-4	Not used
3	Enable interrupt on video blanking (VBL)
2	Enable interrupt on next VBL after button pressed
1	Enable interrupt on next VBL after movement
0	Mouse on/off

Only bits 0 and 3 are used by the Toolkit.

Data is passed to and from the driver in a 5 byte parameter region located at an address returned by the AttachDriver call. The first four bytes are normally mouse position information (except during the ClampMouse call when they are min and max values). The last byte is the mouse button and interrupt status byte defined as follows:

Table J-7. Mouse Status and Interrupt Byte Status Modes

Bit#	Function
7	Button is down
6	Button was down at last reading
5	Mouse moved since last reading
4	Reserved
3	VBL interrupt
2	Button press interrupt
1	Mouse movement interrupt
0	Reserved

Suggestion Box

We do our best to provide you with complete, bug-free software and documentation. With products as complex as compilers and programming utilities, this is difficult to do. If you find any bugs or areas where the documentation is unclear, please let us know. We will do our best to correct the problem in the next revision. We would also like to hear from you if have any comments or suggestions regarding our product.

To help us better understand your comments please use the following form in your correspondence and mail it to: Kyan Software Inc., 1850 Union Street #183, San Francisco, CA 94123.

Name_____
Address_____
City_____State_____ZIP_____
Telephone:
(day)_____ (evening)_____

Kind of Problem
__ Software Bug
__ Documentation Error
__ Suggestions
__ Other _____

Software Description
Product Name _____
Version No. _____
Date Purchased _____

Kyan Software Products You Use
__ Kyan Pascal
__ System. Utilities Toolkit
__ MouseText Toolkit
__ TurtleGraphics Toolkit
__ Kyan Macro Assembler/Linker
__ Advanced Graphics Toolkit
__ MouseGraphics Toolkit
__ Other _____

Your Hardware Configuration
Type/Model of Computer _____
How many and what kind of disk drives _____
What is your screen capability: ___40 Column ___80 Column
How much RAM?____K (what kind of RAM Board?_____)
What kind of printer and interface card do you use?_____
_____
What kind of modem?_____
Other information about your computer system: _____
_____

What do you use this software for?

___ Education (I am a __ teacher __ student)
___ Hobby
___ Professional Software Development
___ Other _____

Problem Description (if appropriate, please include a disk or program listing).

_____
_____
_____
_____
_____
_____
_____
_____
_____

Suggestions

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

MT 8605A