Toolkit I

SYSTEM UTILITIES

USERS MANUAL

KYAN SOFTWARE INC.
SAN FRANCISCO, CALIFORNIA

TOOLKIT I

SYSTEM UTILITIES

Requires
Kyan Pascal (Version 2.0)
and
an Apple II with 64K of memory

TABLE OF CONTENTS

Notice

Use of Routines in this Toolkit

Kyan Software hereby grants you a non-exclusive license to merge or use the routines in this Toolkit in conjunction with your own programs for either private or commercial purposes.

Copyright

This users manual and the computer software (programs) described in it are copyrighted by Kyan Software 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. 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.

This restriction on copies does not apply to copies of individual routines copied and distributed as an integral part of programs developed by the purchaser of this Toolkit.

Backup Copies

We strongly recommend that you make and use backup copies of the Toolkit diskette. Keep your original Kyan diskettes in a safe location in case something happens to your copies. (Remember ..... Murphy is alive and well, and he loves to mess with computers!)

Copy Protection

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

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. This software is licensed "as is". The entire risk as to its quality and performance is with the Buyer. Should the software prove defective following its purchase, the buyer (and not Kyan Software, its distributors, or its retailers) assumes the entire cost of all necessary servicing, repair, or correction and any incidental, or consequential damages.

In no event, will Kyan Software be liable for direct, or indirect, incidental, or consequential damages resulting from any defect in the software even if it has been advised of the possibility of such damages. The sole obligation of Kyan Software Inc. shall be to make available for purchase, modifications or updates made by Kyan Software to the software which are published within one year from date of purchase, provided the customer has returned the registration card delivered with the software.

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.

If any provisions or portions of this Agreement shall be held by a court of competent jurisdiction to be contrary to law, the remaining provisions of this Agreement shall remain in full force and effect. The validity, construction and performance of this Agreement shall be governed by the substantive law of the State of California.

This Agreement constitutes the entire agreement between the parties concerning the subject matter hereof.

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.

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 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 sugges-tions, comments and, yes, even criticisms of our products. 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

A. Introduction

Thank you for purchasing this System Utilities Toolkit. It is designed for use with Kyan Pascal (Version 2.0 or later) and an Apple // with at least 64K of memory (RAM).

Overview

The Toolkit contains many useful and powerful routines which can be merged directly into your Kyan Pascal programs. These routines are grouped into four libraries or directories.

I. ProDOS Utility Library

This library contains routines which provide support for various ProDOS functions and procedures from within Pascal programs.

o Delete	o Rename	o Copy	o SetPrefix
o Append	o Lock	o Unlock	o MakeDir
o RemDir	o Find	o ScanFile	o FileType
o GetDir	o GetPrefix	o Format	o GetTime
o GetDate	o FindClock	o SetDate	o PrtMLIerror
o BSave	o BLoad	o SetTime	o GetTimeM
o PrintFile			

II. Device Driver Library

This library contains functions and procedures which establish communication between your application programs and an external device (i.e., mouse, joystick or trackball).

o FindMouse	o InitMouse	o MouseClick	o MouseHeld
o MouseMoved	o MouseX	o MouseY	o ZerMouse
o SetMouseXY	o SetXBounds	o SetYBounds	o HomeMouse
o EndMouse	o PrtMouseChar	o Button0	o Button1
o JoyStX	o JoyStY		

<u>III. Screen Management Library</u>

This library contains routines used to control screen functions.

o CLS	o GoToXY	o TAB	o Inverse
o Normal	o ScrollUp	o ScrollDown	o ClrLine
o ClrEOLN	o ClrEOP	o Col80	o CursorX
o CursorY	o GetChar	o ScrnTop	o ScrnBottom
o ScrnFull	o IDMachine	o ON40	o ON80

<u>IV. Other System Utilities</u>

o Random Number Routines

-- Seed ("seed" the random number generator routine)
-- Rnd (return a random number between 0 and 1)
-- Random (return a random number in range [min .. max])

o Conversion Routines

-- Integer to String	-- Real Number to String
-- String to Real Number	-- String to Integer

o Line Parsing Routine

o Sort/Merge Routine

How to Use the System Utilities

The routines in each Library are text files and are structured to be used as "include" files in your Pascal programs. To use them:

1. Copy the desired Toolkit routine(s) into your current working directory.
2. Declare the "included" file(s) in the declarations portion of your program.
3. Call the routine(s) as required in the body of your program.

Some libraries require global types to be separately declared. The steps for declaring these global types are described later in this Manual.

While most of the Toolkit routines are independent of all others, some routines incorporate others in the body of their programs. In these circumstances, it is necessary to include both Toolkit routines in your Pascal program. If a routine is dependent on some other routine, the dependency is noted in the application notes for the routine.

It is a good idea to review the section in Chapter III of your Kyan Pascal manual which describes the use of "include" files in your Pascal programs. You should also look at Chapter V which describes assembly language programming and Appendices C-F which list the meaning of MLI and other error messages.

You are encouraged to examine the source code of the Toolkit routines. To do so, simply load the routine's include file using the Kyan Text Editor. The source files are fully commented, and so you should be able to easily follow the logic and flow of the program. You can also modify any of the routines, if desired, and customize them for your particular application.

The Appendix illustrates the directory and file organization of the System Utilities Toolkit disk. Always be sure to specify the complete pathname of the include file when you are copying routines into your working directory or running the demonstration programs. Also, when running the demo programs, be sure there is a copy of the Kyan Pascal Runtime Library (LIB) in the working directory.

Demonstration Programs

The System Utilities Toolkit contains a number of demonstration programs which illustrate the use of Toolkit routines. Most of these programs are included in both source and object code formats.

TITLE	DESCRIPTION
CATALOG.P	This program will print a short catalog of the directory you indicate. It will list each file and its type and size (in blocks). This program demonstrates the ProDOS utilities.
MOUSE.DEMO.P	Uses the Mouse routines in conjunction with a simple menu application interfaced to the mouse. This program demonstrates Device utilities.
RANDOM.DEMO.P	Play a simple random number game. This program demonstrates seeding of the random number generator.
ESORT.DEMO.P	Demonstrates the Sort routine.
MERGE.DEMO.P	Demonstrates the Merge routine.
MOUSETEXT.DEMO	(Object code only). Displays the procedure and complete table of mouse text characters.
TURTLE.DEMO	(Object code only). Illustrates the power of Kyan's TurtleGraphics Toolkit.

B. ProDOS Utility Library

Overview

The ProDOS Utility Library contains 26 different routines. They include a mix of functions and procedures which can be incorporated into your Pascal programs. Each ProDOS utility routine is described on following pages.

The ProDOS routines included in this Library are:

Append	BLoad
BSave	Copy
Delete	Filesize
Filetype	Find
FindClock	Format
GetClock	GetDate
GetDir	GetPrefix
GetTime	Lock
MakeDir	PrintFile
PrtMLIerror	RemDir
Rename	ScanFile
SetClock	SetDate
SetPrefix	SetTime
Unlock	

The ProDOS routines are listed in alphabetical order.

Using the ProDOS Utility Library

To use the ProDOS Utility routines, you must first declare a set of global types and then "include" the desired routine after the variable and type declarations in your Pascal program. (Please refer to Chapter III of the Kyan Pascal User Manual for more information about the use of Include files in Pascal programs.) Once a routine is included, it can be called as often as needed in your program.

Declaring Global Types

You can declare the global variables in either of two ways. First, you can simply type the following global declarations into your Pascal program:

```
Type
  FileString = ARRAY [1..16] of CHAR;
  PathString = ARRAY [1..65] of CHAR;
  FilePointer = ^FileRecord;
  FileRecord = RECORD
         Filename : FileString;
         NextFile : FilePointer
         END;
```

Alternately, you can "include" the file on the program disk called ***PRODOS.TYPES.I*** in your Pascal program using the following format:

```
Type
#i PRODOS.TYPES.I
```

Both methods acheive the objective of declaring the global types used in the ProDOS Utility Library routines.

Notes

1. Don't forget to place a copy of all the files "included" in your Pascal program in the same working directory as the main program. If you forget, the compiler will not be able to find the file and a "File Not Found" compiler error will occur.

2. There are no global types to be declared with Device Driver routines.

3. The Utility program disk contains a set of sample programs. The program file ***CATALOG.P*** demonstrates the ProDOS utilities.

4. All of the ProDOS Utility Library routines are similar in that: (1) each ProDOS Function returns the MLI error code of its operation; and, (2) all pathnames passed must be padded with spaces to the right.

Command Name: ***Append***

Syntax: FUNCTION APPEND(VAR sourcepath, addpath :
PathString) : INTEGER;

Description: Append the contents of "sourcepath" to "addpath". Type checks are not made; the user must do this first using the FILETYPE function included in this toolkit. A 512-byte local buffer is used for the data transfer area.

****

Command Name: ***BinaryLoad***

Syntax: FUNCTION BLOAD (VAR pathname : PathString;
len, dest : INTEGER) : INTEGER;

Description: Load the first "len" bytes of a BINary image named "pathname" starting at "dest". If "len" is zero, the entire file is loaded. **For Example:** To load a hi-resolution graphics image into page 1 of hi-res memory, the command BLOAD(name, 8192, 8192) would be used since the image length is 8192 bytes long ('len') and hi-res page 1` starts at memory location 8192 ($2000) ('dest'). If the image had been saved with a length of 8192 previously, the command BLOAD(name,0,8192) would perform the same opereation since a lentgth specification of zero loads the entire binary file into memory.

****

Command Name: ***BinarySave***

Syntax: FUNCTION BSAVE (VAR pathname : PathString;
len, dest : INTEGER) : INTEGER;

Description: Save a BINary image named "pathname" starting at "dest" of "len" bytes in length. **For example:** To save a hi-resolution graphics image stored in hi-res page 1, the command BSAVE(name, 8192, 8192) would store a binary image of the first 8192 bytes it found (len of 8192) starting at location 8192 ($2000) (start) into file 'name'.

Command Name: ***Copy***

Syntax: FUNCTION COPY(VAR sourcepath, destpath : PathString):
INTEGER;

Description: Copy the file designated by "sourcepath" to the file designeated by "destpath". The destination filename already exists, it is destroyed before the copy begins. COPY uses a 512-byte buffer in which to perform the data transfer. The volumes involved in the copy function must both be on-line at the time of the COPY. COPY will not ask for volumes to be inserted and removed.

****

Command Name: ***Delete***

Syntax: FUNCTION DELETE(VAR pathname:PathString):
INTEGER;

Description: Delete the file designated in "pathname". Error codes are returned as Integers

****

Command Name: ***FileSize***

Syntax: FUNCTION FILESIZE (VAR pathname : PathString;
VAR fsize : INTEGER) : INTEGER;

Description: Set FSIZE to the number of disk blocks occupiedby "pathname" (i.e. return size of file in blocks).

Command Name: ***FileType***

Syntax: FUNCTION FILETYPE (VAR pathname : PathString;
VAR ftype : INTEGER) : INTEGER;

Description: Set FTYPE to the file type of "pathname" (i.e. return type of file).

****

Command Name: ***Find***

Syntax: FUNCTION FIND (VAR filename : PathString;
VAR found : BOOLEAN) : INTEGER;

Description: Returns "found" TRUE if the file name passed is located in the system prefix (Working Directory). Only the Working Directory is searched.

****

Command Name: ***FindClock***

Syntax: FUNCTION FINDCLOCK : INTEGER;

Description: Returns the slot number of a ThunderClock or compatible Apple II clock card. If there is no compatible clock in the system, the number returned is zero. A ThunderClock is recognized by the following identification bytes:

LOCATION	VALUE
Cx00	8
Cx01	$27
Cx02	$28

where "x" is the slot number of the card.

Command Name: ***Format***

Syntax: FUNCTION FORMAT (slot, drive : INTEGER; volname :
FileString; VAR fmterror : INTEGER) : INTEGER;

Description: Format the volume in (slot#, drive#) and name the new volume "volname". If a format error occurs, an MLI error code or one of the following code numbers (fmterror) will be returned

$27 - Disk access error
$33 - Drive too slow
$34 - Drive too fast

If an error occurs during the writing of a boot block or construction of the Volume Bit Map, it is returned as the function value. The volume name specified must conform to pathname guidelines and begin with a '/', or an "INVALID PATHNAME" will be returned as the function value.

Due to the size of the FORMAT routine, it must be loaded from disk each time it is used. A file named FORMAT.OBJ is included in the ProDOS utilities directory on your diskette. This file must be located in the Working Directory when the FORMAT routine is called.

FORMAT.OBJ is BLOADed into memory at location $2000 and requires the use of HiRes graphics page 1. For this reason, any program using the FORMAT function must begin with the following code:

```
#A
_UsesHires
#
```

The "_UsesHires" declaration tells the compiler not to allocate any memory between locations $2000 and $3FFF to any part of the Pascal program or runtime environment. (Please refer to your Kyan Pascal manual for more information).

Declaring a Pascal program to be a "_SystemFile" does not effect the FORMAT routine.

Command Name: ***GetClock***

Syntax: PROCEDURE GETCLOCK (VAR mon, day, yr, hr, min,
weekday, sec, millisec: INTEGER);

Description: Returns readings from Thunderclock. If no clock is present, the values returned are undefined. The FINDCLOCK function must be called previous to this procedure.

****

Command Name: ***GetDate***

Syntax: PROCEDURE GETDATE (VAR day, month, year
: INTEGER);

Description: Returns the day (0..31), month (0..12) and year (00..99) as integers. (**NOTE:** The function FindClock must always be called before a GetDate or SetDate procedure is used. Without this call, these routines will not know there is a clock card in the system.)

****

Command Name: ***GetDirectory***

Syntax: FUNCTION GETDIR (VAR dirname : PathString;
VAR ListPtr : FilePtr) : INTEGER;

Description: Returns a linked list of filenames in "dirname". The list is terminated by a NIL. If the directory is located but empty, LISTPTR returns pointing at NIL. If the MLI return code is non-zero, LISTPTR will be undefined.

Command Name: ***GetPrefix***

Syntax: PROCEDURE GETPREFIX (VAR prefix : PathString) ;

Description: Returns the current system prefix (Working Directory). "Prefix" returns with the first 64 characters the system prefix; the remainder is buffered by spaces. If the system prefix is null, the returned array contains only blanks.

****

Command Name: ***GetTime***

Syntax: PROCEDURE GETTIME (VAR hour, minute: INTEGER);

Description: Returns the system time in hour/minute military format

****

Command Name: ***Lock***

Syntax: FUNCTION LOCK (VAR pathname : PathString) : INTEGER;

Description: Deny Write, Delete, and Rename access to the file designated in the "pathname".

Command Name: ***MakeDirectory***

Syntax: FUNCTION MAKEDIR (VAR dirname : PathString) :
INTEGER;

Description: Create a directory named "dirname". The directory is created as a linked subdirectory type.

****

Command Name: ***PrintFile***

Syntax: PROCEDURE PRINTFILE (pathname : PathString;
Slot, LeftMargin, RightMargin, TopMargin, BottomMargin, CPI:
INTEGER; header : FileString) : INTEGER;

Description: Print the text file designated by "pathname" to the printer in "slot" using margins listed. Header is the printer conditioning control codes. If an "include" file is encountered in the text file, an attempt is made to find and print the included file. If it fails, it is ignored and an MLI code is returned which corresponds to an error caused by the pathname specified and NOT the include files. The printer must be on-line and at top of form when this procedure is called.

****

Command Name: ***PrintMLIerror***

Syntax: PROCEDURE PRTMLIERROR (errorcode : INTEGER);

Description: Print the MLI error code passed at current cursor position. Unknown error codes are printed as "*PRODOS ERROR $xx*" where *xx* is the hexidecimal error code. The error text is printed from the current position of the cursor and NO carriage return is generated by the output routine.

Command Name: ***RemoveDirectory***

Syntax: FUNCTION REMDIR (VAR dirname : PathString) :
INTEGER;

Description: DELETE the directory "dirname", first checking to make sure that "dirname" is an empty directory. If "dirname" is not empty, a "File Access Error" will be returned.

****

Command Name: ***Rename***

Syntax: FUNCTION RENAME (VAR oldpath, newpath : PathString) :
INTEGER;

Description: Rename the file defined by "oldpath" with the name defined by "newpath". The files must be in the same directory for the rename to succeed.

****

Command Name: ***ScanFile***

Syntax: FUNCTION SCANFILE (VAR pathname, string : PathString;
VAR position : INTEGER) : INTEGER;

Description: Scan the TEXT file designated in "pathname" for the string designated in "string". If the string is found, "position" returns the byte number of the file position of the first character in the string which matchs. If no match is found, "position" returns a value of-1. The SCANFILE search is NOT case sensitive. This command is useful for searching identification fields stored in text files (e.g., high score files).

Command Name: ***SetClock***

Syntax: FUNCTION SETCLOCK(Month, Day, Year, Hours, Minutes,
Dayofweek: INTEGER):INTEGER;

Description: Set the ThunderClock peripheral card to the values passed. The ThunderClock or compatible card must be write enabled for the setting to succeed. The function values returned are:

Code	Error Description
0	No error
1	Month not in 0..59
2	Day not in range according to month
3	Year not 86..99
4	Hour4 not in range 0..59

If an error occurs no time change takes place. The FINDCLOCK function must be called previous to this function.

****

Command Name: ***SetDate***

Syntax: FUNCTION SETDATE (day, month, year:
INTEGER): INTEGER;

Description: Set the system date to the value specified by the user. Error codes may be returned which correspond to an out-of-bounds value:

Code	Error Description
0	No error
1	Month is not between 1 and 12
2	Day is out of range (according to month passed)
3	Year is not between 86 and 99

If an error occurs, no date change takes place. (**NOTE:** The function FindClock must always be called before either the SetDate or GetDate procedure is used. Without this call, the these routines will not know there is a clock card in the system.)

Command Name: ***SetPrefix***

Syntax: FUNCTION SETPREFIX (VAR newprefix : PathString) : INTEGER;

Description: Set the system prefix (or Working Directory) to the pathname specified. If the pathname is all blanks, the system prefix is set to the root volume (null)

****

Command Name: ***SetTime***

Syntax: FUNCTION SETTIME (hours, minutes : INTEGER) : INTEGER;

Description: Set the system time to the values passed. If an error occurs, the function values returned are:

Code	Error Description
0	No error
1	Hour is not in range 0..23
2	Minute is not in range 0..59

If an error occurs, no time change takes place.

****

Command Name: ***Unlock***

Syntax: FUNCTION UNLOCK(VAR pathname : PathString) : INTEGER;

Description: Reverse the effects of the LOCK function.

C. Device Driver Library

Overview

The routines in the Device Driver Library allow you to link external devices to your Pascal programs. The routines are intended for use with a mouse, trackball (which behave exactly like a mouse), or joystick.

If you are planning a major project using mouse routines, you should look at Kyan's **MouseText Toolkit**. The routines in this **System Utility Toolkit** are quite primitive in comparison. The MouseText Toolkit provides all of the macros needed for windows, pull-down menus, option selection via mouse, and icon manipulation.

A mouse interface works best when it is "interrupt driven", that is, when the programmer sets up a series of routines which the mouse firmware (i.e., the ROM's on the Mouse card) calls as events occur. The routines in this Toolkit are not interrupt driven. Instead, the program must continuously poll the device status to determine when a condition changes. This method is less efficient than the interrupt technique but is also much easier to use in small Pascal programs. You can look at the Mouse Demo program (described on the next page) to see a practical method for interfacing the mouse with your program.

For a complete discussion of mouse firmware and the interface between mouse firmware and your Apple II, please refer to the documentation which you received with your mouse.

The routines in the Device Driver Library are:

Button0	Button1	EndMouse
FindMouse	HomeMouse	InitMouse
JoyStkX	JoyStkY	MouseClick
MouseHeld	MouseMoved	MouseX
MouseY	PrtMouseChar	SetMouseXY
SetXBounds	SetYBounds	ZerMouse

Using the Device Drive Library

To use the Device Driver routines, you must first "include" the desired routine after the variable and type declarations in your Pascal program. (Please refer to Chapter III of the Kyan Pascal User Manual for more information about the use of Include files in Pascal programs.) Once the routine is included, you can call it as often as needed in your program.

Notes

1. Don't forget to place a copy of all the files "included" in your Pascal program in the same working directory as the main program. If you forget, the compiler will not be able to find the file and a "File Not Found" compiler error will occur.

2. There are no global types to be declared with Device Driver routines.

3. The program disk contains a set of sample programs. The program file Mouse.Demo.P demonstrates the use of mouse routines in conjunction with TurtleGraphics. You can use the Kyan Pascal editor to examine the source code and to see how the device routines can be utilized in your own programs.

4. The use of any mouse routine requires FUNCTION FINDMOUSE to be in the host program. Also, PROCEDURE INITMOUSE must be loaded and called before any other mouse routines are used.

Command Name: ***Button0***

Syntax: FUNCTION BUTTON0 : BOOLEAN;

Description: This function returns the value TRUE if button 0 or the Open-Apple Key is pressed.

****

Command Name: ***Button1***

Syntax: FUNCTION BUTTON1 : BOOLEAN;

Description: This function returns the value TRUE if button 1 or the Closed-Apple Key is pressed.

****

Command Name: ***EndMouse***

Syntax: PROCEDURE ENDMOUSE;

Description: This procedure disables the mouse system interrupts.

Command Name: ***FindMouse***

Syntax: FUNCTION FINDMOUSE : INTEGER;

Description: This function returns the slot number of the mouse card. If none is present, FINDMOUSE will return a zero and all other calls to mouse routines will be ignored. The FINDMOUSE function must be called in every program using mouse routines (along with the INITMOUSE procedure).

****

Command Name: ***HomeMouse***

Syntax: PROCEDURE HOMEMOUSE;

Description: This procedure moves the mouse X,Y coordinates to their lowest boundaries as defined by the SetXBounds and SetYBounds routines.

****

Command Name: ***InitializeMouse***

Syntax: PROCEDURE INITMOUSE;

Description: This procedure prepares the mouse firmware for use. It sets the X, Y lower and upper bounds to 0 and 1023 and disables the mouse firmware interrupts. INITMOUSE must be called before any other mouse routines are used (the FINDMOUSE procedure must also be called as part of initializing the mouse routines).

Command Name: ***JoyStickX***

Syntax: FUNCTION JOYSTX : INTEGER;

Description: This function returns a value between 0 and 255 for the joystick X coordinate (paddle 0).

****

Command Name: ***JoyStickY***

Syntax: FUNCTION JOYSTY : INTEGER;

Description: This function returns a value between 0 and 255 for the joystick Y coordinate (paddle 1).

****

Command Name: ***MouseClick***

Syntax: FUNCTION MOUSECLICK : BOOLEAN;

Description: This function returns the value TRUE is the mouse button is down.

Command Name: ***MouseHeld***

Syntax: FUNCTION MOUSEHELD: BOOLEAN;

Description: This function returns the value TRUE if the mouse button has been down since the last reading of its status.

****

Command Name: ***MouseMoved***

Syntax: FUNCTION MOUSEMOVED : BOOLEAN;

Description: This function returns the value TRUE if the mouse's position has changed since the last reading of its status.

****

Command Name: ***MouseX***

Syntax: FUNCTION MOUSEX : INTEGER;

Description: This function returns the value of the mouse's X coordinate (0 thru 1023).

Command Name: ***MouseY***

Syntax: FUNCTION MOUSEY : INTEGER;

Description: This function returns the value of the mouse's Y coordinate (0 thru 1023).

****

Command Name: ***PrintMouseCharacter***

Syntax: PROCEDURE PRTMOUSECHAR (ch:CHAR);

Description: This procedure prints the character passed as a mouse character at the current cursor position.

****

Command Name: ***SetMouseXY***

Syntax: PROCEDURE SETMOUSEXY (x,y:INTEGER);

Description: This procedure sets the mouse firmware coordinates to the values x and y.

Command Name: ***SetXBounds***

Syntax: PROCEDURE SETXBOUNDS (xmin, xmax : INTEGER);

Description: This procedure sets the upper and lower bounds for the X coordinate.

****

Command Name: ***SetYBounds***

Syntax: PROCEDURE SETYBOUNDS (ymin, ymax : INTEGER);

Description: This procedure sets the upper and lower bounds for the Y coordinate.

****

Command Name: ***ZeroMouse***

Syntax: PROCEDURE ZERMOUSE;

Description: This procedure zeroes the X,Y mouse coordinates on firmware.

D. Screen Management Library

Overview

The Screen Management Library contains 20 different routines. They include a mix of functions and procedures which can be incorporated into your Pascal programs. Each Screen Management routine is described on following pages.

The Screen Management routines included in this Library are:

CLREOLN	(Clear to end of line)
CLREOP	(Clear to end of page)
CLRLINE	(Clear line)
CLS	(Clear screen)
COL80	(Check for 80 column card)
CURSORX	(Return X position of cursor)
CURSORY	(Return Y position of cursor)
GETCHAR	(Return keypress character)
GOTOXY	(Move cursor to coordinates X,Y)
IDMACHINE	(Return machine ID information)
INVERSE	(Set inverse mode)
NORMAL	(Set video to normal)
ON40	(Enable 40 column display)
ON80	(Enable 80 column display)
SCROLLDOWN	(Scroll down 1 line)
SCROLLUP	(Scroll up 1 line)
SCRNBOTTOM	(Set bottom screen margin)
SCRNFULL	(Return display to full size)
SCRNTOP	(Set top screen margin)
TAB	(Move cursor to position X)

Using the Screen Management Library

To use the Screen Management routines, you must first "include" the desired routine after the variable and type declarations in your Pascal program. (Please refer to Chapter III of the Kyan Pascal User Manual for more information about the use of Include files in Pascal programs.) Once the routine is included, you can call the routines as often as needed in your program.

Notes

1. Don't forget to place a copy of all the files "included" in your Pascal program in the same working directory as the main program. If you forget, the compiler will not be able to find the file and a "File Not Found" compiler error will occur.

2. There are no global types to be declared with Screen Management routines.

3. The Screen Management routines use the following convention:

   Cursor X position: 0 thru 39 (0 thru 79 in 80 column mode)
   Cursor Y position: 0 thru 23

Command Name: ***Clear to End of Line***

Syntax: Procedure CLREOLN;

Description: Clear from the cursor to the end of the line.

****

Command Name: ***Clear to End of Page***

Syntax: Procedure CLREOP;

Description: Clear from the cursor to the end of the page.

****

Command Name: ***Clear Line***

Syntax: Procedure CLRLINE;

Description: Clear horizontal line y. Cursor does not move.

****

Command Name: ***Clear Screen***

Syntax: Procedure CLS;

Description: Clear the current text display.

Command Name: ***Column 80***

Syntax: Function COL80 : Boolean;

Description: Returns the value TRUE if the 80 column firmware is active.

****

Command Name: ***Cursor X Position***

Syntax: Function CURSORX : Integer;

Description: Return the X (horizontal) position of the cursor.

****

Command Name: ***Cursor Y Position***

Syntax: Function CURSORY : Integer;

Description: Returns the Y (vertical) position of the cursor.

****

Command Name: ***Get Character***

Syntax: Function GETCHAR : Char;

Description: Wait for a keypress and then return it as a character.

Command Name: ***Go To Position X,Y***

Syntax: Procedure GOTOXY (x,y : integer);

Description: Move the cursor to screen coordinates (X,Y). If the values passed are out of the range of the current screen (for example, an x coordinate of 65 while in forty column mode), the command is ignored.

Notes:
1. Be sure to use this GOTOXY routine and not any previously published versions. This routine automatically recognizes the four different versions of the Apple II and treats the firmware accordingly.

2. GOTOXY (0,0) moves the cursor to the top left corner of the screen.

****

Command Name: ***Identify Hardware Configuration***

Syntax: Procedure IDMACHINE (VAR version : Char; VAR card80,
extend80 : Boolean);

Description: Returns information regarding the hardware configuration as follows:

version:	' ' - Apple]['+' - Apple][+ 'E' - Apple //e 'e' - Apple //e with 65c02 chips 'c' - Apple //c
card80	Returns TRUE if an 80 column card is found
extended80	Returns TRUE if the system has more than 64K of memory.

Command Name: ***Inverse Screen Mode***

Syntax: Procedure INVERSE;

Description: Set the screen to inverse mode. Note that lower case characters do not appear correctly in the 40 column mode.

****

Command Name: ***Normal Screen Mode***

Syntax: Procedure NORMAL;

Description: Return the screen mode to normal.

****

Command Name: ***Enable 40 Column Display***

Syntax: Procedure ON40;

Description: Enable the 40 column display (this procedure disables the firmware in the 80 column card).

****

Command Name: ***Enable 80 Column Display***

Syntax: Procedure ON80;

Description: Enable the 80 column display. If the system does not contain an 80 column card, this command is ignored.

Command Name: ***Scroll Down***

Syntax: Procedure SCROLLDOWN;

Description: Scroll the 80 column display down one line. The cursor position remains unchanged.

****

Command Name: ***Scroll Up***

Syntax: Procedure SCROLLUP;

Description: Scroll the 80 column display up one line. The cursor position remains unchanged.

****

Command Name: ***Screen Bottom***

Syntax: Procedure SCRNBOTTOM (x : INTEGER);

Description: Set the bottom margin of the video screen to a value between 0 and 23. This command effects the scope of the CLS command. The SCRNTOP and SCRNBOTTOM procedures are used to limit video output. Setting SCRNTOP and SCRNBOTTOM to the same value will limit the screen to single vertical line of text. The position of this line is determined by the value passed to the procedures.

Command Name: ***Screen Full***

Syntax: Procedure SCRNFULL;

Description: This procedure cancels the SCRNTOP and SCRNBOTTOM commands and returns the screen to full size.

****

Command Name: ***Screen Top***

Syntax: Procedure SCRNTOP (x : Integer);

Description: Set the top margin of the video screen to a value between 0 and 23. This command effects the scope of the CLS command. The SCRNTOP and SCRNBOTTOM procedures are used to limit video output. Setting SCRNTOP and SCRNBOTTOM to the same value will limit the screen to single vertical line of text. The position of this line is determined by the value passed to the procedures.

****

Command Name: ***Tab***

Syntax: Procedure TAB (x : Integer);

Description: Move the cursor to position X in the current horizontal line. Out of range values are ignored.

E. Other System Utilities

Overview

The Other System Utilities Directory contains the following routines.

- o Random Number Routines

REAL	(Generates a random number between 0 and 1)
RANDOM	(Generates a random integer in range, min..max)
SEED	("Seeds" the random number generator)

- o Conversion Routines

  REAL NUMBER TO STRING
  INTEGER TO STRING
  STRING TO REAL NUMBER
  STRING TO INTEGER

- o Line Parse Routine

- o Sort/Merge Routine

Using Other System Utilities

To use the Other System Utilities, you must first "include" the global type declarations (if any) and the desired routines after the variable and type declarations in your Pascal program. (Please refer to Chapter III of the Kyan Pascal User Manual for more information about the use of Include files in Pascal programs.) Once the routine is included, you can call the routine as often as needed in your program.

Note: Don't forget to place a copy of all the files "included" in your Pascal program in the same working directory as the main program. If you forget, the compiler will not be able to find the file and a "File Not Found" error will occur.

Random Number Routines

There are three routines in this group. They can be used in your Pascal programs to generate random numbers.

There are no global types associated with these routines.

****

Command Name: ***Random Number 1***

Syntax: FUNCTION RND: REAL;

Description: Generates a real random number between 0 and 1.

****

Command Name: ***Random Number 2***

Syntax: FUNCTION RANDOM (min, max : INTEGER) : INTEGER;

Description: Returns a random integer between min and max.

Note: *Random Number 2* utilizes *Random Numbner 1* (FUNCTION RND) in its source code. As a result, you must be certain to include a copy of the *Random Number 1* routine in any programs which use *Random Number 2.*

Command Name: *Seed Random Number*

Syntax: PROCEDURE SEED (seed1 seed2, seed3 seed4: INTEGER);

Description: This routine is used in conjunction with either of the random number generators to "seed" the string of random numbers generated. Using this routine, it is possible to fix the starting value of the random number sequence.

To "seed" the Random Number Generator, you first include the Seed and Random Number procedures in your Pascal program. Then, you specify four integers of your choosing (i.e., seed1, seed2, seed3, seed4). When the program runs, the Random Number Generator takes these four values, inputs them into its polynomial equation, and generates a sequence of random numbers. Everytime the program is run, the Random Number Generator produces the same sequence of random numbers. To change the sequence, you simply change one or more of the seed values.

Conversion Routines

This group contains four conversion routines and one global type file.

The global types can be declared by adding the following lines of code to the declarations portion of your Pascal program or by including the file ***CONV.TYPES.I*** found on the System Utilities disk.

```
STRING6  = ARRAY[1.. 6] OF CHAR;
STRING20 = ARRAY[1..20] OF CHAR;
```

****

Command Name: ***Real to String Conversion***

Syntax: PROCEDURE REALTOSTR (VAR number:REAL; leading,
decpt: INTEGER; VAR result: String20);

Description: This routine returns a STRING20 type in the format indicated by "leading" and "decpt". "Leading" is the number of characters to use for the leading digits in the resulting string. "Decpt" is the number of decimal places allowed for expansion. For example:

```
REALNUM:= 35932.382;
REALTOSTR (REALNUM, 10, 5, ANSWER);
WRITELN (ANSWER);
```

will output: 35932.38200   (note the five leading spaces)

Notes:

1. Extra space must be left for negative signs in the "leading" specification. Also, if you specify "leading" to be zero, scientific notation will be used for output.

2. If the real number is too large to fit into the string, the string returned will be filled with # symbols. Also, if the leading characters fit but the number of decimal places do not, then as many numbers to the right of the decimal point that will fit will be used.

Command Name: ***Integer to String Conversion***

Syntax: PROCEDURE INTTOSTR (number: INTEGER; justify:
CHAR; VAR result: String6);

Description: This routine converts the integer passed into string. A leading minus is used when the value is negative. Justification characters are:

R	Right justify number, buffering to left with spaces
Z	Right justify number, buffering to left with zeros
L	Left justify with spaces to right (default).

Notes:

1. Any unrecognizable justify characters are treated as Left.

2. The justify character passed must be a capital letter.

****

Command Name: ***String to Real Conversion***

Syntax: FUNCTION STRTOREAL (VAR number: String20) : REAL;

Description: This routine converts a string passed to a real number.

Notes:

1. Non-numeric characters are ignored.

2. The first decimal point encountered is used for conversion.

3. Negative numbers are valid if the first character in the string is a "-".

Command Name: ***String to Integer Conversion***

Syntax: FUNCTION STRTOINT (VAR number: String6) : INTEGER;

Description: This routine converts a string passed to an integer.

Notes:

1. All non-numeric characters are treated as zeroes.
2. A leading minus will give the INTEGER a negative value.

Line Parsing Routine

The Line Parsing routine gives you a method by which to read and "parse" parameter inputs to a program. The source of this input can be the keyboard or another Pascal program. The line parser reads the input string in the Apple input buffer (location $200 to $2FF); it then looks for spaces and breaks the string into records (words); next, it puts these records into a linked list; and, finally, it returns a pointer which identifies the location of the first record in the linked list.

To use the line parsing routine, you must first declare certain global types in your Pascal program. You can declare these global types by adding the following code to the global declarations portion of your Pascal program or by including the ***Parse.Types.I*** file found in the Other Utilities directory.

```
String127 = ARRAY [1..127] of CHAR;
StrPointer = ^StrRecord;
StrRecord = RECORD
                StrFound : String127;
                NextStr : StrPointer
            END;
```

****

Command Name: ***Line Parse Routine***

Syntax: FUNCTION PARSELINE : StrPointer;

Description: This routine returns a pointer to a linked list containing the records or words found in the line passed. The records are considered terminated when they are followed by at least one space (blank). If a blank line is passed to PARSELINE, the pointer 'ParseLine' will point to NIL.

Merge and Sort Routines

The merge and sort routines are very handy for organizing your files. The MERGE procedure will combine up to five presorted files into a single file that is in alphabetically and/or numerically ascending or descending order. The SORT procedure will arrange a file of any type of record into alphabetical or numerical order.

Global Declarations

To use one, or both, of the routines in a Pascal program, you must first Include the file "SRTMERG.TYPES.I", which declares the data types for both of the procedures. This include file declares the following:

```
        PATHSTRING = ARRAY [1..65] OF CHAR;
        NAMEARRAY = ARRAY [1..7] OF PATHSTRING;
        FIELD_TYPE  = (ALPHA_FIELD, INTEGER_FIELD,
                       REAL_FIELD);
```

MERGE also requires the declaration of a VARiable of type NAMEARRAY in which pathnames will be stored. You can declare your own or include the file "SRTMERG.VARS.I" into the global VAR section of your program. For convenience sake, we will assume you have included the SRTMERG.VARS.I file and are using MERGENAMES as your global VARiable of type NAMEARRAY.

Using the MERGE Routine

The MERGE procedure takes between two and five ordered data files, sorts records as they are encountered, and produces one large resultant file containing those merged records. You must specify which files are to be used as source, and the name of the 'intermediate' (or temporary) file for the completely merged image. You even have the option to specify a second destination file.

The MERGE procedure is stored in file "MERGE.I." The procedure is declared as follows:

```
PROCEDURE MERGE (VAR MERGENAMES: NAMEARRAY;
        SELECT, FNUM, RLEN, KLEN, OSET, ORDER: INTEGER;
        KTYPE: FIELD_TYPE);
```

The Parameters are:

MERGENAMES: The MERGE procedure permits you to sort/merge up to five data files. The names of these files must be stored sequentially in MERGENAMES[1..5], starting at element 1. MERGENAMES[6] must contain the pathname of a temporary file to which the MERGE procedure will write out the merged file. The pathname in MERGENAMES[6] must be a valid pathname (if it is not, MERGE will fail immediately). MERGENAMES[7] may contain a different name for the resulting data file if you wish, or be left blank, depending on the value of SELECT below.

SELECT: SELECT is an index to array MERGENAMES; it must be in the range of 1 to 7 inclusive. SELECT indicates what filename to use as a destination file for the resulting merge output file. If SELECT has a value between 1 and 5, the corresponding data file pathname will be used to write the merged file to, thus replacing the data file contents. If SELECT is 6, the temporary filename indicated by MERGENAMES will be left the only output as a result of the MERGE call. If SELECT is 7, the pathname stored in MERGENAMES[7] will be used as a final output destination by MERGE.

FNUM: FNUM is the number of data files to be merged together by the MERGE procedure. Think of this number as an index to the last pathname in the MERGENAMES array you want merged. FNUM must have a value between 2 and 5 inclusive.

RLEN: RLEN is the record length in bytes. In general, record length is fairly easy to calculate. For more information on calculating record lengths and storage sizes by types please consult the Assembly Language programming section of your Kyan Pascal User Manual.

KLEN: KLEN is the length of the key record field in bytes. If you are sorting with a key field of either REALs or INTEGERs, KLEN is automatically set according to type (8 for REAL or 2 for INTEGER). However, using a key made up of CHARacters (alpha_field) will cause the comparison of the keys to take place against KLEN number of characters. KLEN cannot be longer than 255 bytes.

OSET: OSET is the byte offset of the first byte of the key field in the record. OSET can be thought of as the number of bytes found in the record before the first byte of the key field. Therefore, if the key field in your record was the first field declared, OSET would be passed as a 0, since there are no bytes before the key field in that record layout.

ORDER: ORDER determines in what fashion the resulting sorted file's records will be stored. If ORDER is negative, the records will be sorted in descending (highest first) order. If ORDER is non-negative (zero or positive), the records will be sorted in ascending order (lowest first).

KTYPE: KTYPE indicates the type of key field you have specified. If you are using a key that is a character or an array of characters, specify ALPHA_FIELD as KTYPE. If you are using INTEGERs, specify INTEGER_FIELD; if sorting against real numbers use REAL_FIELD as KTYPE.

Using the ESORT routine

The ESORT routine requires the following:

1. The SRTMERG.TYPES.I file be included as global types
2. The SRTMERG.VARS.I file be included as global variables
3. The MERGE.I file be included in the Pascal host program previous to the ESORT procedure.

The global VARiables declared in SRTMERG.VAR.I are:

```
FYLE              : PATHSTRING;
MERGENAMES        : NAMEARRAY;
KTYPE             : FIELD_TYPE;
RLEN, OSET,
ORDER, KLEN,
FNUM, SELECT      :INTEGER;
```

Each variable listed must be conditioned before calling the ESORT routine.

ESORT should be used when one data file containing records with key fields must be sorted. This is accomplished by following these steps:

1. Assign the name of the file to be sorted to the global variable FYLE. Note that the name of the file to be sorted cannot be longer than 62 characters (ESORT needs the remaining bytes in order to append file suffixes to the pathname specified)

2. The global variables RLEN, KLEN, OSET, ORDER, and KTYPE must be assigned values corresponding to those explained in the MERGE documentation.

3. Call ESORT (remember - ESORT has NO PARAMETERS!) The parameters in MERGE and the global variables used by ESORT have the same name. However, you should always remember to assign the global variables their correct values before calling ESORT.

F. APPENDIX

UTILITIES DISK DIRECTORY

Volume Name: UTILITY.TOOLKIT

Directory: ProDOS.LIB

Include Files:

PRODOS.TYPES.I (Global Types)
DELETE.I
RENAME.I
COPY.I
SETPREFIX.I
GETPREFIX.I
APPEND.I
LOCK.I
UNLOCK.I
MAKEDIR.I
REMDIR.I
GETDIR.I
FIND.I
SCANFILE.I
FILETYPE.I
BSAVE.I
BLOAD.I
FORMAT.I
GETCLOCK.I
GETTIME.I
GETDATE.I
SETCLOCK.I
SETTIME.I
SETDATE.I
FINDCLOCK.I
PRTMLIERROR.I
PRINTFILE.I

Directory:	DEVICE.LIB	
Include Files:		FINDMOUSE.I
		INITMOUSE.I
		MOUSECLICK.I
		MOUSEHELD.I
		MOUSEMOVED.I
		MOUSEX.I
		MOUSEY.I
		ZEROMOUSE.I
		SETMOUSEXY.I
		SETXBOUNDS.I
		SETYBOUNDS.I
		HOMEMOUSE.I
		ENDMOUSE.I
		PRTMOUSECHAR.I
		BUTTON0.I
		BUTTON1.I
		JOYSTX.I
		JOYSTY.I

DIRECTORY:	SCREEN.LIB	
Include Files:		CLS.I
		GOTOXY.I
		TAB.I
		INVERSE.I
		NORMAL.I
		SCROLLUP.I
		SCROLLDOWN.I
		CLRLINE.I
		CLREOLN.I
		CLREOP.I
		COL80.I
		CURSORX.I
		CURSORY.I
		GETCHAR.I
		SCRNTOP.I
		SCRNBOTTOM.I
		SCRNFULL.I
		IDMACHINE.I
		ON40.I
		ON80.I

Directory: OTHER.LIB

Include Files:

CONV.TYPES.I (Global Types)
REALTOSTR.I
STRTOREAL.I
INTTOSTR.I
STRTOINT.I

SEED.I
RND.I
RANDOM.I

SRTMERG.TYPES.I (Global Types)
SRTMERG.VARS.I (Global Variables)
ESORT.I
MERGE.I

PARSE.TYPES.I
PARSELINE.I

Directory: DEMO.LIB

CATALOG.P (Source Code)
CATALOG (Object Code)

MOUSE.DEMO.P (Source)
MOUSE.DEMO (Object)

RANDOM.DEMO.P (Source)
RANDOM.DEMO (Object)

ESORT.DEMO.P (Source)
ESORT.DEMO (Object)

MERGE.DEMO.P (Source)
MERGE.DEMO (Object)

MOUSETEXT.DEMO (Object)

TURTLE.DEMO (Object)

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	__ Kyan Macro Assembler/Linker
__ System. Utilities Toolkit	__ Advanced Graphics Toolkit
__ MouseText Toolkit	__ MouseGraphics Toolkit
__ TurtleGraphics 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

TI 8605A