Kyan Software Inc. • *1850 Union Street #183* • *San Francisco, CA 94123*

Toolkit I

Atari
SYSTEM UTILITIES

USERS MANUAL

KYAN SOFTWARE, INC.
SAN FRANCISCO, CALIFORNIA

TOOLKIT I

Atari
SYSTEM UTILITIES

Requires
Kyan Pascal (Version 2.0)
and
any Atari with 48K of memory

TABLE OF CONTENTS

SECTION	PAGE
PREFACE	I-3
A. INTRODUCTION	
Overview	I-7
How to Use the System Utilities	I-9
Demonstration Programs	I-10
B. INPUT/OUTPUT UTILITIES LIBRARY	
Overview and Routine Summary	I-11
Using the Utility Library	I-12
Routine Descriptions	I-14
C. SYSTEM FUNCTIONS LIBRARY	
Overview and Routine Summary	I-23
Using the Utility Library	I-24
Routine Descriptions	I-25
D. SCREEN MANAGEMENT LIBRARY	
Overview and Routine Summary	I-33
Using the Library	I-34
Routine Descriptions	I-35
E. OTHER SYSTEM UTILITIES	
Overview	I-45
Using the Library	I-46
Random Number Routines	I-47
Conversion Routines	I-49
Line Parse Routine	I-52
Sort/Merge Routine	I-53
Machine ID Routine	I-56

F. APPENDIX (Disk Directory) I - 57

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.

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

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 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.

Kyan also warrants that this software performs substantially in accordance with the published specification statement, the documentation, and authorized advertising. Kyan, if notified of significant errors within 90 days from the date of purchase, will at its option:

a) correct demonstrable and significant program or documentation errors within a reasonable period of time; or

b) provide the customer with functionally equivalent software; or

c) provide or authorize a refund.

Except for the limited warranty described in the preceding paragraphs, Kyan Software makes no other warranties, either express or implied, with respect to the software, its merchantability or its fitness for any particular purposes.

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 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 suggestions, 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 any Atari with at least 48K 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. Input/Output Utility Library

This library contains routines which provide support for various input/output operations from within Pascal programs.

- o Delete
- o Append
- o Get_Dir
- o Get_Char
- o Help_Key
- o Get_Free_IOCB
- o Rename
- o Lock
- o Fill
- o Paddle
- o Bload
- o Copy
- o Unlock
- o Stick
- o Ptrig
- o XIO
- o Load_Char_Set
- o ScanFile
- o Format
- o Strig
- o Funtion_Key
- o BSave

II. System Functions Utility Library

This library contains functions and procedures which perform a number of various system tasks.

- o Disable_Break
- o Enable_Key_click
- o Disable_Cursor
- o Enable_Fine_Scroll
- o Disable_Screen
- o Enable_IO_Beep
- o Set_Background_Color
- o Normal_Char_Set
- o Click_Speaker
- o Fast_Key_Repeat
- o Normal_Key_Debounce
- o Reboot_On_Reset
- o Enable_Break
- o Disable_Attract
- o Enable_Cursor
- o Disable_Keyboard
- o Enable_Screen
- o Set_Border_Color
- o Set_Char_Luminance
- o Activate_Char_Set
- o Slow_Key_Repeat
- o Slow_Key_Debounce
- o Fast_Key_Debounce
- o Disable_Key_click
- o Enable_Attract
- o Disable_Fine_Scroll
- o Enable_Keyboard
- o Disable_IO_Beep
- o Intl_Char_Set
- o Invert_Char_Set
- o Beep
- o Normal_Key_Repeat
- o Atari_Char_Set
- o Freeze_On_Reset

III. Screen Management Library

This library contains routines used to access and control screen functions.

- Clear_Screen
- Clear_To_EOP
- Backspace
- GoTo_Screen_Bottom
- Cursor_Left
- Cursor_Down
- Cursor_X
- Tab
- GotoXY_In_Text_Window
- Clear_To_EOLn
- Insert_Char
- Delete_Char
- GoTo_Screen_Top
- Cursor_Right
- Set_Left_Margin
- Cursor_Y
- Set_Tab
- Clear_Ln_At
- Insert_Line
- Delete_Line
- GoToXY
- Cursor_Up
- Set_Right_Margin
- Write_Escape_Char
- Clear_Tab

IV. Other System Utilities

This library contains a variety of routines which can be useful in developing applications programs.

- 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])
  - Random_Byte
- Conversion Routines
  - Integer to String
  - String to Real Number
  - Real Number to String
  - String to Integer
- ParseLine Routine
- Sort/Merge Routines
  - Esort
  - Merge
- ID Machine (identify hardware configuration)

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) onto your disk.
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 the Appendices which list the meaning of I/O 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 file organization of the System Utilities Toolkit disk. Always be sure to specify the complete filename of the include files or demonstration programs which you are using them. Also, when you are running the demo programs, be sure there is a copy of the Kyan Pascal Runtime Library (LIB) on your disk.

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.

SCANDEMO.P	LOADDEMO.P	RANDEMO.P
BDEMO.P	ESORTD.P	LOADDEMO.P
GETDEMO.P	MERGED.P	COPYDEMO.P
BIGDEMO.P	PARSED.P	CONVERT.P

Important Note: All routine "include" files are located on Side1 of the Atari System Utilties diskette. All demonstration programs are located on Side 2.

B. Input/Output Utility Library

Overview

The Input/Output Utility Library contains 23 different routines. They include a mix of functions and procedures which can be incorporated into your Pascal programs to perform a variety of I/O operations. Each utility routine is described on following pages.

The I/O routines included in this Library are:

Append
BLoad
Copy
Fill
Function_Key
Get_Dir
Get_Free_IOCB
Lock
PaddleTrigger(Ptrig)
ScanFile
StickTrigger(Strig)
XIO
Add_Device
BSave
Delete
Format
Get_Char
Help_Key
Load_Char_Set
Paddle
Rename
Stick
Unlock

Using the Input/Output Utility Library

To use the Input/Output 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
  PathString = ARRAY [1..20] of CHAR;
  ElemPtr = ^ElementRec;
  ElementRec = RECORD
        Entry : ARRAY [1..17] of CHAR;
        Next : ElemPtr
        END;
```

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

```
TYPE
#i IOTypes.i
```

Both methods achieve the objective of declaring the global types used in the I/O Utility Library routines.

Notes

1. Don't forget to place a copy of all the "include" files to be used in your Pascal program on the same disk 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.

4. Many of the I/O functions return an Integer that reflects the Central Input/Output (CIO) status byte. This number corresponds to the error numbers found in Appendix D of your Kyan Pascal Users Manual. Remember, if a one (1) is returned, it means that everything is OK and no error has occurred.

5. Many of the I/O routines need to use the Add_Device procedure, so you must include AddDev.I in your program for them. Any routine that uses a filename must call up Add_Device, so including it is not necesasary for routines such as Stick and Help_key

Command Name: ***Add_Device***

Command Syntax: Add_Device(Filename1, Filename2);

Procedure Syntax: PROCEDURE Add_Device(VAR InStr,
OutStr: PathString);

Description: A string passed to this procedure will be changed so that it is a valid filename by adding the default device as the prefix and/or changing the first letter to a capital.

****

Command Name: ***Append***

Command Syntax: Status:=Append(SourceFilename,
DestFilename);

Function Syntax: FUNCTION Append(VAR Source, Dest:
PathString): Integer;

Description: Append the contents of "Source_Filename" to "Dest_Filename". The value returned will be the CIO status byte.

****

Command Name: ***Binary Load***

Command Syntax: Status := BLoad
(Source_Filename,Length,Start);

Function Syntax: FUNCTION BLoad (VAR Pathname:
PathString; Len, Dest : Integer): Integer;

Description: Load the first "length" bytes of a BINary image named "Source_Filename" starting at memory location "start". If "length" is zero, the entire file is loaded. The value returned will be the CIO status byte.

Command Name: ***Binary Save***

Command Syntax: Status:= BSave(Dest_Filename,
Length,Start);

Function Syntax: FUNCTION BSave (VAR PathName: Path;
Len, Dest: Integer): Integer;

Description: Save a BINary image named "DestFilename" at the memory location starting at address "Start", which is "Length" bytes in length. The value returned will be the CIO status byte.

****

Command Name: ***Copy***

Command Syntax: Status := Copy(Source_Filename,
Dest_Filename)

Function Syntax: FUNCTION COPY(VAR Source, Dest:
PathString): INTEGER;

Description: Copy the file designated by "Source" to the file designated by "Dest_Filename". If the destination filename already exists, it isdestroyed before the copy begins. The value returned is the CIOstatus byte.

****

Command Name: ***Delete***

Command Syntax: Status:= Delete(DestFilename);

Function Syntax: FUNCTION Delete(VAR dest: PathString):
Integer;

Description: Delete the file designated in "DestFilename" from the disk. The value returned will be the CIO status byte.

Command Name: ***Fill***

Command Syntax: Fill(Hor, Ver, Color);

Procedure Syntax: Procedure Fill(X, Y, C: Integer);

Description: Fill a portion of the graphics screen at X,Y with the specified color.

****

Command Name: ***Format***

Command Syntax: Status:= Format (Num, Den);

Function Syntax: FUNCTION Format (DriveNum: Integer;
Density: char): Integer;

Description: Format the disk which is specified by "Num", with (s)ingle or enhance(d) density specified by "Den". The value returned is the CIO status byte.

****

Command Name: ***Function Key***

Command Syntax: Function_Key(Which, Time);

Procedure Syntax: PROCEDURE Function_Key(VAR Result:
PathString; Delay: Integer);

Description: The variable parameter "Which" will be changed so that it reflects which of the three function keys are pressed during the period of time specified by the Integer "Time".

Command Name: ***Get Free IOCB***

Command Syntax: Free_IOCB := Get_Free_IOCB;

Function Syntax: FUNCTION Get_Free_IOCB: Integer;

Description: Search for an IOCB not in use and return its number.

****

Command Name: ***Get Directory***

Command Syntax: Status:= Get_Dir (Source_Filename,
File_Pointer);

Function Syntax: FUNCTION Get_Dir (VAR DirName: PathString;
Header: ElemPtr) : Integer;

Description: Get the directory of the disk drive specified by "Source_Filename". For example,

Status:= GetDir('D1:*.* ', FilePointer);

would request a directory for drive one, and

Status:= GetDir('D2:*.* ', FilePointer);

would indicate drive two. The filenames are put in a linked list, which is *terminated* by a NIL. If the directory is empty, "FilePointer" returns pointing at NIL. The value returned by the function is that of the CIO status byte.

Command Name: ***Help _Key***

Command Syntax: Help_Key(Which);

Procedure Syntax: PROCEDURE Help_Key(VAR Out: PathString);

Description: Change the VARiable "Which" to reflect which keyboard and help key combination has been pressed, if any.

****

Command Name: ***Load Character Set***

Command Syntax: Load_Char_Set (Source_Filename; MemLoc);

Procedure Syntax: PROCEDURE Load_Char_Set (VAR FN:Pathstring; VAR Address: Integer);

Description: Load a 1K character set into memory and change the variable MemLoc to the starting address where the set is stored.

****

Command Name: ***Lock***

Command Syntax: Status:= Lock(Dest_Filename);

Function Syntax: FUNCTION Lock (VAR Fylename: PathString): Integer;

Description: Deny Write, Delete, and Rename access to the file designated by "Dest_Filename". Status returns the CIO status byte.

Command Name: ***Get Character***

Command Syntax: Status := Get_Char(Ch);

Function Syntax: FUNCTION Get_Char (VAR Ch: Char): Integer;

Description: Get a character from the keyboard and change the variable "Ch" so that it holds this character. Status returns the CIO status byte.

****

Command Name: ***Paddle***

Command Syntax: Number := Paddle_Num;

Function Syntax: FUNCTION Paddle(Num: Integer): Integer;

Description: Returns the value held by the potentiometer, or paddle, specified by Num, which is 0-7.

****

Command Name: ***Paddle Trigger***

Command Syntax: Pressed:= Ptrig(Paddle_Num);

Function Syntax: FUNCTION Ptrig(Num: Integer):Boolean;

Description: Returns TRUE if paddle trigger is pressed where Paddle_Num equals 0 through 7.

Command Name: ***Rename***

Command Syntax: Status:= Rename(Old_Filename, New_Filename);

Function Syntax: FUNCTION Rename (VAR OldName, NewName: PathString): Integer;

Description: Rename the file defined by "Old_Filename" with the name defined by "New_Filename". The value returned will be the CIO status byte.

****

Command Name: ***Scan File***

Command Syntax: ScanFile (Source_Filename, ScanString, Pos);

Procedure Syntax: PROCEDURE ScanFile (VAR Path: PathString; VAR Target: SearchType; VAR Position: Integer)

Description: Scan the TEXT file designated by "Source_Filename" for the string of characters designated in "String". If the string is found, "Pos" is changed to reflect the byte number of the file position of the first character in the string which matchs. If no match is found, "Pos" 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: ***Stick***

Command Syntax: Number := Stick_Num;

Function Syntax: FUNCTION Stick(Num: Integer): Integer;

Description: Returns the value that reflects the position of the joystick specified by "Stick_Num", which is 0-3.

Command Name: ***Stick Trigger***

Command Syntax: Pressed:= Strig(Stick_Num);

Function Syntax: FUNCTION Strig(Num: Integer):Boolean;

Description: Returns TRUE if the JoyStick button is pressed where Stick_Num equals 0 through 3.

****

Command Name: ***Unlock***

Command Syntax: Status:= Unlock(Dest_Filename);

Function Syntax: FUNCTION Unlock(VAR Fylename:
PathString): Integer;

Description: Reverse the effects of the LOCK function. Value returned is CIO status byte.

****

Command Name: ***XIO***

Command Syntax: Status := XIO(Command, IOCB, Auxillary1,
Auxillary2,Filename);

Function Syntax: FUNCTION XIO(cmd, iocb, aux1, aux2: Integer;
VAR Fyle: Path String): Integer;

Description: Sets up Extended Input/Output for CIO. The value returned is CIO status byte.

C. System Functions Library

Overview

The System Functions Library contains 34 different routines. They include a mix of functions and procedures which can be incorporated into your Pascal programs to perform a variety of system functions. Each routine is described on the following pages.

The system functions routines included in this library are:

Attract	(Enable and Disable)
Beep	
Break	(Enable and Disable)
Character Set -- Activate	
Character Set -- Atari	
Character Set -- International	
Character Set -- Normal	
Character Set -- Inverted	
Click Speaker	
Cursor	(Enable and Disable)
Fine Scroll	(Enable and Disable)
IO Beep	(Enable and Disable)
Keyboard	(Enable and Disable)
Key Click	(Enable and Disable)
Key Debounce	(Slow, Normal, and Fast)
Key Repeat	(Slow, Normal, and Fast)
Reset -- Freeze on	
Reset -- Reboot on	
Screen	(Enable and Disable)
Set Background Color	
Set Border Color	
Set Character Luminance	

Using the System Functions Library

To use the System Function 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 on the same disk 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 Special Functions Library routines.

3. The program disk contains a set of sample programs. These programs demonstrate the use of System routines. 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.

Command Name: ***Attract -- Enable and Disable***

Command Syntax: Enable_Attract;
Disable_Attract;

Procedure Syntax: PROCEDURE Enable_Attract;

Description: Enable sets the "Attract Mode", which cycles screen colors.

****

Command Name: ***Beep***

Command Syntax: Beep;

Procedure Syntax: PROCEDURE Beep;

Description: Produces a beep from the speaker. This rude noise is the same as pressing CONTROL -2.

****

Command Name: ***Break -- Enable and Disable***

Command Syntax: Disable_Break;
Enable_Break;

Procedure Syntax: PROCEDURE Disable_Break;

Description: This procedure disables the BREAK Key so that it will not stop the program flow when pressed.

Command Name: ***Character Set -- Activate***

Command Syntax: Activate_Char_Set(MemLoc);

Procedure Syntax: PROCEDURE Activate_Char_Set(ADR: Integer);

Description: Tell the computer to use a previously loaded character set. The parameter is the MSB of the starting memory location, which is returned by the Load_Char_Set procedure.

****

Command Name: ***Character Set -- Atari***

Command Syntax: Atari_Char_Set;

Procedure Syntax: PROCEDURE Atari_Char_Set;

Description: Tell the computer to use the Standard Atari Character Set built in to ROM.

****

Command Name: ***Character Set -- International***

Command Syntax: Intl_Char_Set;

Procedure Syntax: PROCEDURE Intl_Char_Set;

Description: Tell the computer to use the International Character Set built in to Atari XL/XE machines. **Note: XE/XL models only.**

Command Name: ***Character Set -- Normal and Inverted***

Command Syntax: Invert_Char_Set;
Normal_Char_Set;

Procedure Syntax: PROCEDURE Invert_Char_Set;

Description: Invert the character set (i.e.,flip it upside-down). Normal returns the characters to right-side-up.

****

Command Name: ***Click Speaker***

Command Syntax: Click_Speaker;

Procedure Syntax: PROCEDURE Click_Speaker;

Description: Click the monitor speaker.

****

Command Name: ***Cursor -- Enable and Disable***

Command Syntax: Disable_Cursor;
Enable_Cursor;

Procedure Syntax: PROCEDURE Disable_Cursor;

Description: Disable the visibility of the cursor.

Command Name: ***Fine Scroll -- Enable and Disable***

Command Syntax: Enable_Fine_Scroll;
Disable_Fine_Scroll;

Procedure Syntax: PROCEDURE Enable_Fine_Scroll;

Description: Enable fine scroll so that lines do not appear "choppy" when scrolling vertically. **Note: XE/XL models only.**

****

Command Name: ***IO Beep -- Enable and Disable***

Command Syntax: Disable_IO_Beep;
Enable_IO_Beep;

Procedure Syntax: PROCEDURE Disable_IO_Beep;

Description: Disable the beeping made during input/output operations.

****

Command Name: ***Keyboard -- Enable and Disable***

Command Syntax: Disable_Keyboard;
Enable_Keyboard;

Procedure Syntax: PROCEDURE Disable_Keyboard;

Description: Disable the keyboard so that it will not accept input.
Note: XE/XL models only.

Command Name: ***Key Click -- Enable and Disable***

Command Syntax: Disable_Key_Click;
Enable_Key_Click;

Procedure Syntax: PROCEDURE Disable_Key_Click;

Description: Disable the click, or chirp, made when a key is pressed.
Note: XE/XL models only.

****

Command Name: ***Key Debounce -- Slow, Normal, and Fast***

Command Syntax: Slow_Key_Debounce;
Normal_Key_Debounce;
Fast_Key_Debounce;

Procedure Syntax: PROCEDURE Slow_Key_Debounce;

Description: Set the debounce time of the keyboard.
Note: XE/XL models only.

****

Command Name: ***Key Repeat -- Slow, Normal, and Fast***

Command Syntax: Slow_Key_Repeat;
Normal_Key_Repeat;
Fast_Key_Repeat;

Procedure Syntax: PROCEDURE Slow_Key_Repeat;

Description: Set the auto repeat rate for the keyboard.
Note: XE/XL models only.

Command Name: ***Reset -- Freeze on***

Command Syntax: Freeze_On_Reset;

Procedure Syntax: PROCEDURE Freeze_On_Reset;

Description: If the system reset key is pressed after this procedure is called, the computer will lock up.

****

Command Name: ***Reset -- Reboot on***

Command Syntax: Reboot_On_Reset;

Procedure Syntax: PROCEDURE Reboot_On_Reset;

Description: If the system reset key is pressed after this procedure is called, the computer will attempt to re-boot DOS.

****

Command Name: ***Screen Disable***

Command Syntax: Num := Disable_Screen;

Function Syntax: FUNCTION Disable_Screen: Integer;

Description: Turn off the screen so that processing speed will increase by 30%. The number returned should be used to enable the screen.

Command Name: ***Screen -- Enable***

Command Syntax: Enable_Screen(Loc);

Procedure Syntax: PROCEDURE Enable_Screen(Num: Integer);

Description: Enable the screen after it has been disabled. The number "Loc" passed is the one returned by the Disable_Screen function.

****

Command Name: ***Set Background Color***

Command Syntax: Set_Background_Color(Num);

Procedure Syntax: PROCEDURE Set_Background_Color(Col: Integer);

Description: Num is a number from 0 to 255, which controls the background color of the screen.

****

Command Name: ***Set Border Color***

Command Syntax: Set_Border_Color(Num);

Procedure Syntax: PROCEDURE Set_Border_Color(Col: Integer);

Description: Num is a number from 0 to 255, which controls the border color of the screen.

Command Name: ***Set Character Luminance***

Command Syntax: Set_Char_Luminance(Num);

Procedure Syntax: PROCEDURE Set_Char_Luminance(Col: Integer);

Description: Set the brightness of the characters on the Graphics 0 (text) screen. The number is from 0 to 255.

****

D. Screen Management Library

Overview

The Screen Management Library contains 25 different routines which perform various tasks with the screen in the 40 column text mode (Graphics 0). Each Screen Management routine is described on following pages.

The Screen Management routines included in this Library are:

Backspace
Clear_Ln_At (Clear line at)
Clear_To_EOLn (Clear to end of line)
Clear_To_EOP (Clear to end of page)
Clear_Screen
Clear_Tab
Cursor_Down (Move cursor down 1 line)
Cursor_Left (Move cursor left 1 character)
Cursor_Right (Move cursor right 1 character)
Cursor_Up (Move cursor up 1 line)
Cursor_X (Return X position of cursor)
Cursor_Y (Return Y position of cursor)
Delete_Char (Delete Character)
Delete_Line
GoTo_Screen_Top
GoTo_Screen_Bottom
GoToXY (Move cursor to coordinates X,Y)
GoToXY_In_Text_Window
Insert_Char (Insert Character)
Insert_Line
Set_Left_Margin
Set_Right_Margin
Set_Tab
Tab (Move cursor to position X)
Write_Escape_Char

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 copy all the files "included" in your Pascal program onto the disk that your program will be compiled from. If you forget, the compiler will not be able to find the file and a "File Not Found" error message will appear.

2. There are no global types to be declared with Screen Management routines nor do they use any other include files.

3. The Screen Management routines use the following convention for cursor position on the screen:

Horizontal position:	0 minimum (left-most column) 39 maximum (right-most column)
Vertical position:	0 minimum (top-most row) 23 maximum (bottom-most row).

Command Name: ***Backspace***

Command Syntax: Backspace;

Procedure Syntax: PROCEDURE Backspace;

Description: Delete the character behind the cursor, moving the cursor to its position. This has the same effect as pressing DELETE/BACKSPACE.

****

Command Name: ***Clear Line at Y***

Command Syntax: Clear_Line_at (Vert);

Procedure Syntax: PROCEDURE Clear_Line_at (Y: Integer);

Description: Clear line at given cursor position specified by Vert. The position of the cursor does not change.

****

Command Name: ***Clear Screen***

Command Syntax: Clear_Screen;

Procedure Syntax: PROCEDURE Clear_Screen;

Description: Clear all text from the screen and put the cursor at the top of the page. The same effect as pressing CONTROL-CLEAR.

Command Name: ***Clear Tab***

Command Syntax: Clear_Tab;

Procedure Syntax: PROCEDURE Clear_Tab;

Description: Clear the tab stop at the current cursor position. This has the same effect as pressing CONTROL-TAB.

****

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

Command Syntax: Clear_To_EOLn;

Procedure Syntax: PROCEDURE Clear_To_EOLn;

Description: Clear text from the current cursor position to the end of the line.

****

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

Command Syntax: Clear_To_EOP;

Procedure Syntax: PROCEDURE Clear_To_EOP;

Description: Clear text from the cursor position to the end of the page or to the right-most position at the bottom of the screen. The cursor position remains the same.

Command Name: ***Cursor X***

Command Syntax: HorPos := Cursor_X;

Function Syntax: FUNCTION Cursor_X: Integer;

Description: Return the current horizontal position of the cursor.

****

Command Name: ***Cursor Y***

Command Syntax: VerPos := Cursor_Y;

Function Syntax: FUNCTION Cursor_Y: Integer;

Description: Return the current vertical position of the cursor.

****

Command Name: ***DeleteCharacter***

Command Syntax: Delete_Char;

Procedure Syntax: PROCEDURE Delete_Char;

Description: Delete the character on which the cursor is positioned and pull the following text back one character. This has the same effect as pressing CONTROL-DELETE.

Command Name: ***Delete Line***

Command Syntax: Delete_Line;

Procedure Syntax: PROCEDURE Delete_Line;

Description: Delete all text on the line on which the cursor is positioned. This has the same effect as pressing SHIFT-DELETE.

****

Command Name: ***Go to Bottom of Screen***

Command Syntax: Goto_Screen_Bottom;

Procedure Syntax: PROCEDURE Goto_Screen_Bottom;

Description: Position the cursor at the bottom of the screen in the right-most column.

****

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

Command Syntax: GoToXY(x,y);

Procedure Syntax: PROCEDURE GoToXY(Hor, Vert: Integer);

Description: Move the cursor to screen coordinates (X,Y).

Command Name: ***Position Text in Window***

Command Syntax: GotoXY_In_Text_Window(X,Y);

Procedure Syntax: PROCEDURE GotoXY_In_Text_Window
(HX,VY: Integer);

Description: Same as GoToXY, except it works in the text window of a graphics screen.

****

Command Name: ***Go to Top of Screen***

Command Syntax: Goto_Screen_Top;

Procedure Syntax: PROCEDURE Goto_Screen_Top;

Description: Position the cursor at the top of the screen in the left-most position.

****

Command Name: ***Insert Character***

Command Syntax: Insert_Char;

Procedure Syntax: PROCEDURE Insert_Char;

Description: Insert one character (space) at the current cursor position, moving the following text forward one character. This has the same effect as pressing CONTROL-INSERT.

Command Name: ***Insert Line***

Command Syntax: Insert_Line;

Procedure Syntax: PROCEDURE Insert_Line;

Description: Insert an entire line at the current cursor position, moving the following lines of text down one line. This has the same effect as pressing SHIFT-INSERT.

****

Command Name: ***Move Cursor Down***

Command Syntax: Cursor_Down;

Procedure Syntax: PROCEDURE Cursor_Down;

Description: Move the cursor one character downward. This has the same effect as pressing CONTROL-DOWN ARROW.

****

Command Name: ***Move Cursor Left***

Command Syntax: Cursor_Left;

Procedure Syntax: PROCEDURE Cursor_Left;

Description: Move the cursor one character to the left. This has the same effect as pressing CONTROL-LEFT ARROW.

Command Name: ***Move Cursor Right***

Command Syntax: Cursor_Right;

Procedure Syntax: PROCEDURE Cursor_Right;

Description: Move the cursor one character to the right. This has the same effect as pressing CONTROL-RIGHT ARROW.

****

Command Name: ***Move Cursor Up***

Command Syntax: Cursor_Up;

Procedure Syntax: PROCEDURE Cursor_Up;

Description: Move the cursor one character upward. This has the same effect as pressing CONTROL-UP ARROW.

****

Command Name: ***Set Left Margin***

Command Syntax: Set_Left_Margin (Margin_Value);

Procedure Syntax: PROCEDURE Set_Left_Margin(Num: Integer);

Description: Set the left margin of the video screen to the value specified.

Command Name: ***Set Right Margin***

Command Syntax: Set_Right_Margin (Margin_Value);

Procedure Syntax: PROCEDURE Set_Right_Margin (Num: Integer);

Description: Set the right margin of the video screen to the value specified.

****

Command Name: ***Set Tab***

Command Syntax: Set_Tab;

Procedure Syntax: PROCEDURE Set_Tab;

Description: Set a tab stop at the current cursor position. This has th same effect as pressing SHIFT-TAB.

****

Command Name: ***Tab***

Command Syntax: Tab;

Procedure Syntax: PROCEDURE Tab;

Description: Tab the cursor to the next tab stop. This has the same effect as pressing the TAB key.

Command Name: ***Write Escape Character***

Command Syntax: Write_Escape_Char;

Procedure Syntax: PROCEDURE Write_Escape_Character;

Description: Write the escape character to the screen. The next character written may appear as a special graphic character. This also is useful for sending control codes to the printer.

E. Other System Utilities

Overview

The final System Utilities library contains the following routines.

- Random Number Routines

  Rnd (Generates a random number between 0 and 1)
  Random (Generates a random integer in range, min..max)
  Seed ("Seeds" the random number generator)
  Random_Byte (Generates a random integer, 0 .. 255)

- Conversion Routines

  RealToStr
  IntToStr
  StrToReal
  StrToInt

- Parse Line Routine

- Sort/Merge Routine

  ESORT
  MERGE

- ID Machine 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 on the same disk 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 four 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***

Command Syntax: Num := Rnd;

Function Syntax: FUNCTION Rnd: Real;

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

****

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

Command Syntax: Num := Random;

Function Syntax: FUNCTION Random (min, max : Integer) : Integer;

Description: Returns a random integer between min and max.

Note: *Random Number 2* utilizes *Random Number 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***

Command Syntax: Seed(Num1, Num2, Num3,Num4);

Procedure 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.

****

Command Name: ***Random Number 3***

Command Syntax: Num := Random_Byte

Function Syntax: FUNCTION Random_Byte: Integer;

Description: This routine returns a random integer in the range zero to 255.

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 ***ConvTypes.i*** found on the System Utilities disk.

```
String5  = ARRAY[1..5] OF Char;
String6  = ARRAY[1.. 6] OF Char;
String20 = ARRAY[1..20] OF Char;
```

****

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

Command Syntax: RealToStr(Num, Lead_Spcs, Dec_Pt, Final);

Procedure 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 does not, then as many numbers to the right of the decimal point that will fit will be used.

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

Command Syntax: IntToStr(Num, Just, Final);

Procedure Syntax: PROCEDURE InToStr (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***

Command Syntax: Num := StrToReal(Num_Str);

Function 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***

Command Syntax: Num := StrToInt(Num_Str);

Function 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 Atari input buffer; 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 ***ParseT.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***

Command Syntax: Ptr := ParseLine;

Function 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 "SrtMergT.I", which declares the data types for both of the procedures. This include file declares the following:

```
PATHSTRING = ARRAY [1..20] 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 filenames 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 filename of a temporary file to which the MERGE procedure will write out the merged file. The filename in MERGENAMES[6] must be a valid filename (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 filename 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 filename 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 filename 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 SrtMrgT.I file be included as global types
2. The SrtMrgV.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 SrtMrgV.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.

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.

****

Command Name: ***Identify Machine Type***

Command Syntax: Mach_Type := ID_Machine;

FunctionSyntax: ID_Machine: Char;

Description: Return a character that represents the version of the ROM chips in the machine. "A" indicates old ROMs and "B" indicates the latest version. If it is not known, it will return a '?'.

F. APPENDIX

UTILITIES DISK DIRECTORY

Disk Side 1

A. Input/Output Library

IOTYPES.I (Global Types)

Append.I
BLoad.I
BSave.I
Copy.I
Delete.I
Format.I
FunctKey.I
GetChar.I
GetDir.I
Lock.I
Paddle.I (Paddle, Ptrig)
Rename.I
ScanFile.I
Stick.I (Stick, Strig)
Unlock.I
HelpKey.I
LoadCSet.I
Fill.i
XIO.I

B. System Functions Library

Break.I	(Enable and Disable)
KeyClick.I	(Enable and Disable)
Attract.I	(Enable and Disable)
Cursor.I	(Enable and Disable)
FinScrol.I	(Enable and Disable)
Keyboard.I	(Enable and Disable)
Screen.I	(Enable and Disable)
IOBeep.I	(Enable and Disable)
Cset.I	(Invert, International, Normal)
ActCset.I	
Beep.I	
Click.I	
Repeat.I	(Slow, Normal, Fast)
Debounce.I	(Slow, Normal, Fast)
Reset.I	(Freeze, Reboot)
Colors.I	(File with color routines)
Border.I	(Set Border Color)
BkGround.I	(Set Background Color)
Lumin.I	(Set Character Luminance)

C. Screen Management Library

Clears.I	(Clear_Screen, Clear_To_EOLn, Clear_Ln, Clear_To_EOP)
Inserts.I	(Insert_Char, Insert_Line)
Deletes.I	(Backspace, Delete_Char, Delete_Line)
Gotos.I	(GoToXY, GoTo_Screen_Top, GoTo_Screen_Bottom)
Cursors.I	(Cursor_Left, Cursor_Right, Cursor_Up, Cursor_Down)
Margins.I	(Set_Left_Margin, Set_Right_Margin)
CursorPs.I	(Cursor_X, Cursor_Y)
GoToXYTW.I	(GoToXY_In_Text_Window)
PosInTW.I	(Pos_In_Text_Window)
Tabs.I	(Tab, Set_Tab, Clear_Tab)
ESCChar.I	(Write_Escape_Char)

D. Other System Utilties

Convtyps.I	(Global Types)
Esort.I	
Merge.I	
Randoms.I	(Random, Seed, Rnd, RandomByte)
RtoS.I	
StoR.I	
ItoS.I	
StoI.I	
ParseLn.I	
IDMach.I	

Disk Side 2

Demonstration Programs

BDEMO.P (Source)
BDEMO (Object)

BIGDEMO.P (Source)
BIGDEMO (Object)

CONVERT.P (Source)
CONVERT (Object)

COPYDEMO.P (Source)
COPYDEMO (Object)

ESORTD.P (Source)
ESORTD (Object)

GETDEMO.P (Source)
GETDEMO (Object)

LOADDEMO.P (Source)
LOADDEMO (Object)

MERGED.P (Source)
MERGED (Object)

PARSED.P (Source)
PARSED (Object)

RANDEMO.P (Source)
RANDEMO (Object)

SCANDEMO.P (Source)
SCANDEMO (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
__ System. Utilities Toolkit
__ Other _____
__ Kyan Macro Assembler/Linker
__ Advanced Graphics Toolkit

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 printer and interface 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

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