Apple III

Pascal Technical Reference Manual

Supplement

Acknowledgements

The Apple III Pascal system is based on UCSD Pascal. "UCSD PASCAL" is a trademark of the Regents of the University of California. Use thereof in conjunction with any goods or services is authorized by specific license only and is an indication that the associated product or service has met quality assurance standards prescribed by the University. Any unauthorized use thereof is contrary to the laws of the State of California.

Contents

1

SOS Calls From Pascal

The Apple III Sophisticated Operating System (SOS) has a generalized command set to access attached devices, manage memory and the event mechanism, and take advantage of other features of the machine. This supplement describes a set of routines, SOSIO, available on the disk that came with this manual. These routines allow direct access to SOS calls from a Pascal program.

By making direct system calls to SOS, you can expand the potentially limiting nature of Pascal's input and output (I/O) constructs. You can also take advantage of SOS device independence and of the Apple III's large memory. As you can see in some of the examples in Chapter 6 of the *Pascal Technical Reference Manual*, using SOS calls you can determine the memory available on the system and allocate space beyond the 64K bytes permitted by the Pascal stack-heap. Other calls may be used to manage the SOS event system, deal with the system clock, and restart the system. Detailed descriptions of the SOS calls may be found in the *SOS Reference Manual*.

Contents of the Disk

Two forms of the Pascal interface to SOS are provided on the enclosed disk. One provides a set of Pascal procedures; the other is similar, but provides the routines as Pascal functions returning TRUE if the call succeeds (with a return code of zero) and FALSE if it fails (a non-zero return code). The function interface always returns TRUE for SOS calls that do not generate a return code.

The function version is the one used in the examples and has the unit name SOSIO. A linked code file, SOSIO.CODE, is included on the disk. The procedure version has the unit name PSOSIO.

You may find it worthwhile to include only the routines you need. Thus, source code is provided. To conserve space in the compiled and linked unit as well as on the disk, comments are omitted in the Pascal source code for the units. These "interface only" units are named ISOSIO.TEXT and IPSOSIO.TEXT. A listing of the fully commented function version of the unit is included in this supplement.

The disk also has the source code for the assembly language routines that actually make the SOS calls. The main file that must be assembled is ASMSOS.TEXT. Note that the file ASMSOS has a flag that tells whether the version being assembled is the function or procedure version.

To build a function version of SOSIO, set FUNCTION to be Ø1 in ASMSOS and assemble it. Compile ISOSIO. Link the files with the ISOSIO object code as the host file and the ASMSOS object code as the only library file. To build a procedure version, PSOSIO, set FUNCTION to ØØ in ASMSOS before assembling and compile IPSOSIO.

The disk also includes the source code to two programs printed in Chapter 6 of the *Pascal Technical Reference Manual*. DEVTRAN.TEXT is the program that translates between Pascal unit numbers and SOS device names and numbers. Compile it (with SOSIO present) to a file DEVTRAN.CODE. Then, either install SOSIO into the SYSTEM.LIBRARY, or make SOSIO the program library by naming it DEVTRAN.LIB.

Two files make up the program illustrating the use of SOS Extended Memory: XMOVE.TEXT contains the assembly language; USEXMOVE.TEXT contains the Pascal source code. Assemble XMOVE to XMOVE.CODE; compile USEXMOVE to USEXMOVE.CODE (with SOSIO.CODE present). Link the two files with USEXMOVE as the host file and XMOVE as the library file. Again, SOSIO.CODE must be either placed in the SYSTEM.LIBRARY or made into a program library for the program to work.

SOS File Calls

Table 1 lists the SOS file management calls. The following paragraphs describe some possible uses of these calls from Pascal programs. Note that the routines described here are executable only on the Apple III using SOS-formatted disks, even though Apple III Pascal allows you to read UCSD Pascal-formatted disks as well as edit, compile, and link source text that was created on the Apple II UCSD Pascal system.

Table 1. SOS File Management Calls

Call	**Description**
SOS_Create	Creates a file on a block device and preallocates blocks for a new file, if desired.
SOS_Destroy	Removes a file from a blocked device.
SOS_Rename	Changes the name of a file.
SOS_Set_Info	Defines the directory information to be associated with a specified file.
SOS_Get_Info	Returns the directory information of a file.
SOS_Volume	Returns the volume name, total blocks in use, and total number of free blocks for any block device.
SOS_Set_Prefix	Sets the system prefix pathname. This is *not* the Pascal prefix.
SOS_Get_Prefix	Returns the current system prefix pathname.
SOS_Open	Opens any SOS file, including devices such as the console.

Table 1. SOS File Management Calls (cont.)

Call	Description
SOS_New_Line	Disables or enables and sets the "read until" character for the specified SOS file.
SOS_Read	Reads from a specified file.
SOS_S_Read	Reads from a specified file into an indexed buffer. Useful for reading in a Pascal string variable after turning off range checking. The same as SOS_Read except that the buffer read into is indexed by a specified number of bytes.
SOS_Write	Writes to a specified file.
SOS_S_Write	Writes to a specified file from an indexed buffer. Useful for writing string variables.
SOS_Close	Closes the specified file. If the passed reference number is Ø, all user files are closed.
SOS_Flush	Writes out any information currently buffered by SOS to the specified file. Works in a similar fashion to SOS_Close when a Ø reference number is passed to it. This gives the applications programmer the ability on demand to write out to disk all SOS file buffers.
SOS_Get_B_Mark	Gets the current file mark rounded up to the closest block number.

Table 1. SOS File Management Calls (cont.)

Call	Description
SOS_Set_B_Mark	Sets the current file position to the passed block number.
SOS_Get_B_EOF	Gets the current EOF rounded up to the closest block number.
SOS_Set_B_EOF	Sets the EOF to the passed block number.
SOS_Get_Mark	Gets the current file mark and returns the low 16 bits in "Low" and the high order 8 bits (of the 24 bit mark) in "Hi."
SOS_Set_Mark	Sets the mark to the 24 bit quantity passed.
SOS_Get_EOF	Gets the current EOF and returns the 24 bit quantity in "Low" and "Hi."
SOS_Set_EOF	Sets the EOF to the 24 bit quantity passed.
SOS_Get_Lev	Returns the value of the system file level. All SOS_Opens assign this level to files opened until it is changed. All SOS_Close and SOS_Flush operations on multiple files will operate only on those files that were opened with a level greater than or equal to the current level.
SOS_Set_Lev	Changes the current value of the system file level.

The Console

The Apple III text mode is supported by the SOS console driver. Accessing the console driver directly rather than using Pascal read and write routines, you can mix commands to the driver (described in the *Standard Device Drivers Manual*) along with text. With a single call to SOS, you can turn off the cursor, clear the screen, position the cursor at any X,Y position, write a line of data, set a viewport, scroll newly written text up one line, and turn on the cursor. (A viewport is an arbitrary rectangular area that can be defined by a program.) Because the contents of any viewport can be saved and then restored, error messages can temporarily overlay other information; the previous data can then be restored in a single command request to the console driver.

Writing to a Printer

With Apple III Pascal, data is passed to a printer on a character-by-character basis. This means that data passed to a printing device by a Pascal write or unitwrite statement is broken down into many SOS calls. This takes many times more SOS overhead than a direct request to a SOS printer driver. The performance improvement you get by using a SOS_ Open and then SOS_ Write calls to write to a printer is clear.

Apple III Pascal uses 1100 bytes (550 words) of buffer space in the data space for each file that is opened, including any character device such as a printer or the console. A buffer is thus expected as a parameter to SOS_ Open. However, because SOS uses this buffer only with block devices, a dummy variable may be passed as the buffer parameter when accessing character devices. This could save stack and heap space when you use direct SOS calls to output to a printer or other character device, compared to the amount of space used by Pascal I/O procedures.

On a block device, the buffer area passed as a parameter to SOS_ Open must not be touched by other parts of the program and must always be available while the file is open. Because of this requirement, the usual scope rules for Pascal must be rigorously followed to keep Pascal from deallocating a SOS file buffer before the file is closed. Failing to do this will result in a fatal SOS error, SYSTEM FAILURE $0F, "Invalid buffer number."

The file type, modification date and time, and other attributes can be modified for every SOS file by means of a SOS_ Set_ Info call. Since SOSIO calls communicate directly with SOS, no Pascal preprocessing to convert control characters, such as the DLE and CONTROL-C, takes place on input or output. (See the section on Device I/O in the *Pascal Programmer's Manual, Volume 1*.) This reduces the amount of processing and ensures that you write out any control characters you intended.

SOS Device Calls

Table 2 lists the SOS device calls. Given a SOS device name, you can use SOS_ Get_ D_ Num to translate the name into a SOS device number, which may be used as a parameter to other SOS calls. SOS_ D_ Info may be used to get information concerning configured devices. SOS_ D_ Status and SOS_ D_ Control may be used to get status information concerning a device and to issue control calls. Consult the *Standard Device Drivers Manual* or the descriptions of other device drivers for parameters for specific devices.

Table 2. SOS Device Calls

Call	**Description**
SOS_D_Status	Issues a device status request and returns the results for the specified device.
SOS_D_Control	Issues a device control request to any SOS device.
SOS_Get_D_Num	Returns the SOS device number for the named device.
SOS_D_Info	Returns device information for the specified device, if it is configured.

SOS Memory Management, Utility, and Additional Calls

Table 3 lists the SOS memory management and utility calls, as well as some additional routines. You have already seen—in Chapter 6 of the *Pascal Technical Reference Manual*—examples of the use of memory management calls to take advantage of the large memory available on the Apple III.

Table 3. SOS Memory Management, Utility, and Additional Calls

Call	**Description**
SOS_Request_Seg	Requests a certain type and size of memory segment.
SOS_Find_Seg	Finds a memory segment of a certain type, if possible. If not, the size of the largest free segment is returned along with an error indication.
SOS_Change_Seg	Changes the starting and ending addresses of the segment by adding or releasing the specified number of pages, if possible. If not, an error indication is returned along with the maximum allowable page count.
SOS_G_Seg_Info	Returns the beginning and ending locations, size in pages, and identification code of the specified segment.
SOS_G_Seg_Numb	Returns the segment number of the segment, if any, containing the specified address.
SOS_Rel_Seg	Releases the segment with the specified segment number, if any.

Table 3. SOS Memory Management Utility, and Additional Calls (cont.)

Call	Description
SOS_S_Fence	Sets the value of the user event fence to the specified value.
SOS_G_Fence	Returns the value of the user event fence.
SOS_Set_Time	Sets the system date/time and resets the clock chip, if present.
SOS_Get_Time	Returns the current system date/time.
SOS_Get_Analog	Reads the analog and digital inputs from an Apple III Joystick connected to port A or B on the back of the Apple III.
SOS_Terminate	Terminates the currently executing program and interpreter.
Up_Load	Not a SOS call. Uploads the SOS character set from $CØØ-$FFF into the passed buffer.
At_Sign	Not a SOS call. Returns the address of a Pascal variable as an integer.

The SOS utility calls deal with the system clock/calendar, the event fence, the analog input ports, and other system resources.

Additional routines are provided to Up_ Load the current character set; an application may then download a new character set and, upon termination, restore the original with a console SOS_ D_ Control call with a control code 16.

Some Hints

Because Apple III Pascal allows assembly language reference (VAR) parameters to bypass type-checking, you can use this feature to allow greater flexibility in defining output parameter data types. For example, any Pascal data type can be passed as the argument to any reference parameter in the assembly language routines described here. This lets the assembly language routine overwrite whatever variable has been passed as a call by a reference parameter. A possible side effect is that you can easily pass the wrong variable to any reference parameter and have the assembly language routine place the returned value in whatever data type you've just passed it. This could cause your program to crash.

In the Apple III Pascal system, a segment that contains external (that is, assembly language) routines is not allowed to cross any boundary between 32K-byte banks. Therefore, the Pascal system may leave unusable holes in memory when loading units containing assembly language routines. You may avoid this by making use of an intrinsic unit version of the SOS Calls.

When compiling your main program, use the "{$NOLOAD+}" compile-time option as the first statement. Then add a "{$R SOSIO}" compile-time command after the main procedure's BEGIN statement. The Pascal interpreter will load the SOS unit first, allowing your P-code only Pascal program to cross any bank boundary encountered further along in the loading process.

Although the SOS calls are presented as Apple III Pascal intrinsic units, the declarations can be incorporated as needed by suffixing an "External;" after each function or procedure declaration and compiling them along with any Pascal main program. However, a SOS_ Data area must always be included because all the routines use it as the SOS parameter block-data area. Additionally, various assembly language routines must be used in pairs, since they share code. They are:

SOS_ D_ Status and SOS_ D_ Control
SOS_ Set_ Info and SOS_ Get_ Info
SOS_ Read and SOS_ S_ Read
SOS_ Write and SOS_ S_ Write
SOS_ Get_ B_ Mark and SOS_ Get_ B_ EOF
SOS_ Set_ B_ Mark and SOS_ Set_ B_ EOF
SOS_ Get_ Mark and SOS_ Get_ EOF
SOS_ Set_ Mark and SOS_ Set_ EOF

Linking the Pascal program with the necessary assembled external routines would allow it to be run without having a SYSTEM.LIBRARY or program library on-line.

Conclusion

By using SOSIO in your Pascal programs, you will gain flexibility and power in addition to a possible performance improvement.

The use of SOSIO can also result in significant memory savings. The complete set of routines when assembled as Pascal procedures is:

Device calls	Ø.2K
File calls	1.4K
Utility calls	Ø.2K
Memory calls	Ø.3K
Additional calls	Ø.1K
Data	Ø.1K
Total	2.3K

These totals may be reduced through selective assembly, if necessary. Note that the complete set of file-processing routines is slightly larger than 1K bytes of code. For each Pascal file opened by calling Reset or Rewrite, a mandatory 1K-byte buffer is reserved. Therefore, the entire SOS_ IO file package fits into the space saved using the package to write to just one nonblocked device. Through judicious use of the memory management routines, you can also gain access to far greater memory than would be possible on the Pascal system alone.

2

SOSIO Pascal Code

This section contains a fully commented listing of the Pascal function version of the SOSIO Pascal code.

```
                    SOS Interface--February 1, 1983           Version 1.1

{$CC Copyright 1981, 1982, 1983 Apple Computer Inc.}

{$SETC Intrinsic_Unit   := TRUE}

{ File SOSIO }

{Function interface version of SOSIO:

If the SOS call generates a return code, the value of the SOSIO function is
TRUE if the return code is Ø; it is FALSE if the return code is non-zero.

If the SOS call does not generate a return code, the value of the SOSIO
function is always TRUE.}

Unit SOSIO;

{$IFC Intrinsic_Unit}

Intrinsic CODE 58;

{$ENDC}

Interface

 { Set to TRUE to compile SOS calls for: }

 {$SETC SOS_File_IO      := TRUE}
 {$SETC SOS_Device_IO    := TRUE}
 {$SETC SOS_Utility_IO   := TRUE}
 {$SETC SOS_Memory_Mgt   := TRUE}
 {$SETC SOS_Plus_IO      := TRUE}
```

```
                    SOS Device Calls--February 1, 1983          Version 1.1

{ Device management SOS calls }

{$IFC SOS_Device_IO}

Function SOS_D_Status ( DevNumb, StatusCode : Integer;
                        Var StatusList, RetCode ) : Boolean;

{  Issues a device status request and returns the status from any SOS device. }

{  Input Values :

     DevNumb    : The SOS device number to perform the status/control request.
                  This is obtained through the SOS_Get_D_Num call.
     StatusCode : The status code to be performed.

   Output Values :

     StatusList : A buffer in which to return a status request list.  The
                  length of the buffer is device dependent.
     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_D_Control ( DevNumb, ControlCode : Integer;
                         Var ControlList, RetCode ) : Boolean;

{  Issues a device control request to any SOS device. }

{  Input Values :

     DevNumb    : The SOS device number to perform the status/control request.
                  This is obtained through the SOS_Get_D_Num call.
     ControlCode: The control code of the action to be performed.
     ControlList: The optional control list for the specific control code.

   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Get_D_Num ( Var DevName, DevNumb, RetCode ) : Boolean;

{  Returns the SOS device number for a configured device. }

{  Input Values :

     DevName    : A Pascal string with a maximum length of 15 characters
                  that contains the device name.

   Output Values :

     DevNumb    : The SOS device number is returned as on integer value
                  from 1 to 18.
```

```
                    SOS Device Calls--February 1, 1983          Version 1.1

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_D_Info ( DevNo : Integer; Var DevName, DevList, RetCode )
                    : Boolean;

{  Returns device information (for configured devices), for the passed device
   number. }

{  Input Values :

     DevNo      : A SOS device number from Ø to 18.  All other numbers
                  are INVALID.

   Output Values :

     DevName    : The SOS device name corresponding to the passed DevNo.
     DevList    : The 11 byte SOS device information list:

                       Byte Ø       : Slot number
                       Byte 1       : Unit number
                       Byte 2       : Device type
                       Byte 3       : Device subtype
                       Byte 4       : Reserved
                       Bytes 5 & 6 : Blocks available
                       Bytes 7 & 8 : Manufacturing ID
                       Bytes 9 & 1Ø: Version number
                   (Use a packed array [Ø..1Ø] of char or Ø..255)

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}
{$ENDC}
```

```
{ File management SOS calls }

{$IFC SOS_File_IO}
Function SOS_Create ( Var Pathname; FileID, AuxID, Storage, EOFBlk : Integer;
                      Var RetCode ) : Boolean;

{  Creates a file on a block device with the specified pathname. }

{  Input Values :

     Pathname   : A Pascal string that is a valid SOS pathname.
     FileID     : The SOS file identification code to associate with the
                  created file.
     AuxID      : The SOS auxiliary identification code.
     Storage    : The storage type to create. One is a standard file, thirteen
                  is a subdirectory file.
     EOFBlk     : The number of blocks to preallocate for the file on a block
                  device.  The range is Ø to 32767 blocks.
   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Destroy ( Var Pathname, RetCode ) : Boolean;

{  Deletes the file specified by the passed pathname. }

{  Input Values :

     Pathname   : The pathname of the file to destroy.

   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Rename ( Var OldPath, NewPath, RetCode ) : Boolean;

{  Renames the OldPath to the NewPath-name. }

{  Input Values :

     OldPath    : A Pascal string pathname to change FROM.
     NewPath    : A Pascal string pathname to change TO.

   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Set_Info ( Var Pathname, FileList; ListLeng : Integer;
                  Var Retcode) : Boolean;

{  Sets the file information specified by the passed pathname and ListLeng. }

{  Input Values :

     PathName   : The pathname to set the file information.
     File List  : The up to 15 byte list (the length SOS uses is determined by
                  ListLeng:
       Byte Ø   - The file attribute bits.  Bit 7 set is destroy OK; bit
                  6 set is rename OK; bit 1 set is write OK; bit Ø is read OK.
       Byte 1   - The file identification code.
       Bytes 2&3- The auxiliary identification code.
       Bytes 11 to
             14 - The packed values for the date and time stamp: Year (Ø..99),
                  Month (1..12), Day (1..31), Hour (1..24), Minute (1..6Ø);
                  stored in four bytes in the following fashion.

                            (high byte)        (low byte)
                      7  6  5  4  3  2  1  Ø | 7  6  5  4  3  2  1  Ø
                     +-----------------------------------------------+
                 Date |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
                     +-----------------------------------------------+
                     |    Year             | Month  |      Day      |

                            (high byte)        (low byte)
                      7  6  5  4  3  2  1  Ø | 7  6  5  4  3  2  1  Ø
                     +-----------------------------------------------+
                 Time |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
                     +-----------------------------------------------+
                     |       Hour            |        Minute          |

     ListLeng   : The file attributes to change.  One is only FileAttr, three
                  is through FileID, fourteen is through AuxID, and fifteen
                  is everything.

   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Get_Info ( Var Pathname, FileList; ListLeng : Integer;
                         Var RetCode ) : Boolean;

{  Gets the file information specified by the passed pathname. }

{  Input Values :

     PathName   : The pathname of the file to get the information from.

     ListLeng   : The length of the file information list to be returned
                  by SOS (as per the FileList definition).
   Output Values :

     FileList   : The file information returned on the file with the pathname
                  passed:

                  Byte Ø         : File attribute
                  Byte 1         : File identification
                  Byte 2 & 3     : (Low,High) Auxiliary identification
                  Byte 4         : Storage type
                  Bytes 5..8     : (Low,High) EOF in bytes
                  Bytes 9 & 1Ø   : Blocks currently used
                  Bytes 11..14   : (Low,High) Modification date and time

                  (Use packed array [Ø..14] of char or Ø..255)

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Volume ( Var DevName, VolName, TotalBlks, FreeBlks, RetCode )
                    : Boolean;

{  Gets volume information on the device specified by the passed DevName. }

{  Input Values :

     DevName    : A Pascal string containing the device name; maximum of
                  15 characters in length.

   Output Values :

     VolName    : The SOS volume name returned in a Pascal string 15 bytes
                  long.
     TotalBlks  : The total number of blocks on the volume, returned
                  as an UNSIGNED integer value (Ø to 65535).
     FreeBlks   : The number of available blocks on the volume, returned
                  as an UNSIGNED integer value (Ø to 65535).
     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Set_Prefix ( Var Prefix, RetCode ) : Boolean;

{  Sets the system prefix (NOT the Pascal prefix!) to the passed prefix
   string. }

{  Input Values :

     Prefix     : A Pascal string up up to 255 characters long containing
                  the system prefix value.  Note that a "/" is automatically
                  added to the end of the system prefix.

   Output Values :
    RetCode    : An integer to contain the SOS return code (a zero means no
                 errors).
}

Function SOS_Get_Prefix ( Var Prefix; Length : Integer; Var RetCode )
                        : Boolean;

{  Gets the current system prefix (NOT the Pascal prefix!). }

{  Input Values :

    Prefix     : A Pascal string[n] to receive the current system prefix.
    Length     : The maximum length of the string.

   Output Values :

    RetCode    : An integer to contain the SOS return code (a zero means no
                 errors).
}

Function SOS_Open ( Var Path; ReqType, Pages : Integer; Var SysBuf, RefNumb,
                    RetCode ) : Boolean;

{  Opens a file with the specified pathname. }

{  Input Values :

    Path       : A Pascal string containing the pathname of the file to
                 be opened.
    ReqType    : The manner in which to open the file, e.g. Ø = file's
                 attribute, 1 = read only, 2 = write only, 3 = read/write.
    Pages      : The number of user supplied pages pointed to by the SysBuf
                 parameter.  Note that passing a Ø means that SOS finds its
                 own buffer.  The maximum value is 4; each page is 256 bytes
                 long.  If a Ø is passed, then SOS ignores the SysBuf
                 parameter, and finds its own buffer.
    SysBuf     : This must be a 1Ø24 byte buffer for SOS to use for the
                 duration of the open. CAUTION : You cannot use OR deallocate
                 this buffer while the file is open.  Use a packed array
                 [Ø..1Ø23] of char.  If the file being opened is not on
                 a blocked device (e.g. a printer, the console), the SysBuf
                 pointer is ignored by SOS.

   Output Values :

    RefNumb    : This is the SOS file reference number returned as an integer
                 value, to be used in SOS_Read's and SOS_Write's to the file.
    RetCode    : An integer to contain the SOS return code (a zero means no
                 errors).
}

Function SOS_New_Line ( RefNumb, Flag : Integer; NewCh : Char; Var RetCode )
                      : Boolean;
```

```
{ Enables/disables the "newline" read mode (i.e. stops a read on the specified
  NewCh when enabled). }

{ Input Values :

    RefNumb     : The reference number of the file.

    Flag        : Ø..127 is disable; 128..256 is enable the newline mode.

    NewCh       : the character to be used as a newline character (terminates
                  the read).

  Output Values :

    RetCode    : An integer to contain the SOS return code (a zero means no
                 errors).
}

Function SOS_Read ( RefNumb : Integer; Var InputBuf; BytesReq : Integer;
                    Var BytesRead, RetCode ) : Boolean;

{ Reads from the file specified by reference number. }

{ Input Values :

    RefNumb    : The reference number returned from the successful SOS_Open
                 request.
    BytesReq   : The number of bytes to read as an UNSIGNED integer value
                 (Ø..65535).

  Output Values :

    InputBuf   : The buffer to read into. Use a packed array [Ø..??] of char.
    BytesRead  : The actual number of bytes read into InputBuf.
    RetCode    : An integer to contain the SOS return code (a zero means no
                 errors).
}

Function SOS_S_Read ( RefNumb : Integer; Var InputBuf; OffSet,
                      BytesReq :Integer; Var BytesRead, RetCode ) : Boolean;

{ The Same as SOS_Read, except that the buffer read into is indexed by OffSet
  bytes (e.g. for a read into a string).
}

Function SOS_Write ( RefNumb : Integer; Var OutputBuf; NumbBytes : Integer;
                     Var RetCode ) : Boolean;

{ Writes to the file specified by reference number. }

{ Input Values :

    RefNumb    : The SOS reference number returned from the successful
```

```
                    SOS File Calls--February 1, 1983          Version 1.1

                SOS_Open request.
      OutputBuf : The Pascal buffer to write to the file, a packed array should
                  be used.
      NumbBytes : The number of bytes to write from BufPtr.

    Output Values :

      RetCode   : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_S_Write ( RefNumb : Integer; Var OutputBuf; OffSet,
                       NumbBytes : Integer; Var RetCode ) : Boolean;

{  The same as SOS_Write, except that the write buffer is indexed by OffSet
   bytes.}

Function SOS_Close ( RefNumb : Integer; Var RetCode ) : Boolean;

{  Closes the file specified by reference number. }

{  Input Values :

      RefNumb   : The reference number returned from the SOS_Open request.

   Output Values :

      RetCode   : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Flush ( RefNumb : Integer; Var RetCode ) : Boolean;

{  The SOS output buffer associated with the file specified by the passed
   reference number is immediately written to the file. }

{  Input Values :

      RefNumb   : The reference number returned from the SOS_Open request.

   Output Values :

      RetCode   : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Get_B_Mark ( RefNumb : Integer; Var BlockNumb, RetCode )
                        : Boolean;

{  Gets the current mark, or position of the file specified by the passed
   reference number, rounded up to the nearest 512 byte block. }

{  Input Values :
      RefNumb    : The SOS file reference number returned by the SOS_Open
                   request.

   Output Values :

      BlockNumb  : The mark rounded up to the nearest 512 byte block number.
                   Use an integer for the Ø..32767 value range.
      RetCode    : An integer to contain the SOS return code (a zero means no
                   errors).
}


Function SOS_Get_B_EOF ( RefNumb : Integer; Var BlockNumb, RetCode )
                        : Boolean;

{  Gets the current EOF of the file specified by the passed reference number,
   rounded up to the nearest 512 byte block. }

{  Input Values :

      RefNumb    : The SOS file reference number returned by the SOS_Open
                   request.

   Output Values :

      BlockNumb  : The EOF rounded up to the nearest 512 byte block number.
                   Use an integer for the Ø..32767 value range.
      RetCode    : An integer to contain the SOS return code (a zero means no
                   errors).
}

Function SOS_Set_B_Mark ( RefNumb, Base, BlockNumb : Integer; Var RetCode )
                         : Boolean;

{  Sets the current mark of the specified file to the 512 byte block number
   specified. }

{  Input Values :

      RefNumb    : The SOS file reference number returned by the SOS_Open
                   request.
      Base       : Where to set the mark relative to : Ø = beginning of the
                   file; 1 = end of the file; 2 = positive from the current
                   position; 3 = negative from the current position.
      BlockNumb  : A integer block number from Ø to 32767 to set the mark to.

   Output Values :

      RetCode    : An integer to contain the SOS return code (a zero means no
                   errors).
}

Function SOS_Set_B_EOF ( RefNumb, Base, BlockNumb : Integer; Var RetCode )
                        : Boolean;
```

```
{  Sets the current EOF of the specified file to the 512 byte block number
   specified. }

{  Input Values :

     RefNumb    : The SOS file reference number returned by the SOS_Open
                  request.
     Base       : Where to set the mark relative to : Ø = beginning of the
                  file; 1 = end of the file; 2 = positive from the current
                  position; 3 = negative from the current position.
     BlockNumb  : A integer block number from Ø to 32767 to set the EOF to.

   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Get_Mark ( RefNumb : Integer; Var Low, Hi, RetCode ) : Boolean;

{  Gets the mark of the specified file as the byte quantity passed as two
   UNSIGNED 16 bit integers. }

{  Input Values :

     RefNumb    : The file reference number returned from the SOS_Open request.

   Output Values :

     Low,Hi     : The mark returned as a 24 bit UNSIGNED quantity.
     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Get_EOF ( RefNumb : Integer; Var Low, Hi, RetCode ) : Boolean;

{  Gets the EOF of the specified file as the byte quantity passed as two
   UNSIGNED 16 bit integers. }

{  Input Values :

     RefNumb    : The file reference number returned from the SOS_Open request.

   Output Values :

     Low,Hi     : The EOF returned as a 24 bit UNSIGNED quantity.
     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}


Function SOS_Set_Mark ( RefNumb, Base, Low, Hi : Integer; Var RetCode )
                      : Boolean;
```

```
{  Sets the mark of the specified file to the byte quantity passed as two
   UNSIGNED 16 bit integers. }

{  Input Values :

     RefNumb    : The file reference number returned from the SOS_Open request.
     Base       : Where to set the mark relative to : Ø = beginning of the
                  file; 1 = end of the file; 2 = positive from the current
                  position; 3 = negative from the current position.
     Low,Hi     : The mark as a 24 bit UNSIGNED quantity.  The high byte
                  of "Hi" MUST BE Ø (i.e. Hi = Ø..255).

   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}
```

__Function__ SOS_Set_EOF (RefNumb, Base, Low, Hi : Integer; __Var__ RetCode)
: Boolean;

```
{  Sets the EOF of the specified file to the byte quantity passed as two
   UNSIGNED 16 bit integers. }

{  Input Values :

     RefNumb    : The file reference number returned from the SOS_Open request.
     Base       : Where to set the mark relative to : Ø = beginning of the
                  file; 1 = end of the file; 2 = positive from the current
                  position; 3 = negative from the current position.
     Low,Hi     : The EOF as a 24 bit UNSIGNED quantity.  The high byte
                  of "Hi" MUST BE Ø (i.e. Hi = Ø..255).

   Output Values :

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}
```

__Function__ SOS_Set_Lev (Level : Integer; __Var__ RetCode : Integer) : Boolean;

```
{  Sets the user event fence priority number. }

{  Input Value :

     Level      : The new file level (valid input is 1..3).

   Output Value:

     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}
```

```
                SOS File Calls--February 1, 1983              Version 1.1

Function SOS_Get_Lev ( Var Level : Integer ) : Boolean;

{  Gets the current file level. }

{  Output Value :

     Level      : The current file level.
}
{$ENDC}
```

SOS Utility Calls--February 1, 1983 Version 1.1

```
{ Utility management SOS calls }

{$IFC SOS_Utility_IO}
Function SOS_S_Fence ( Priority : Integer ) : Boolean;

{  Sets the user event fence priority number. }

{  Input Value :

     Priority   : The new user event fence priority (Ø..255).
}

Function SOS_G_Fence ( Var Priority : Integer ) : Boolean;

{  Gets the current user event fence priority number. }

{  Output Value :

     Priority   : The current user event fence priority.
}

Function SOS_Set_Time ( Var Time ) : Boolean;

{  Sets the system date/time, and resets the clock chip if one is present. }

{  Input Value:

     Time       : An 18 character Pascal string to set the system date and
                  time.  Note that SOS 1.1 keeps the date and time over
                  system cold boots when the clock chip is NOT present.  All
                  SOS files are automatically date/time stamped with this
                  setting.  The string order is:

                 yyyy    mm     dd      w          hh     nn    ss      uuu

                (year) (month) (day) (day of week) (hour) (min.) (sec.) (msec.)
}

Function SOS_Get_Time ( Var Time ) : Boolean;

{  Gets the current system date/time. }

{  Output Value :

     Time       : The current time is returned as an 18 character Pascal
                  string of ASCII digits (see above ordering).
}

Function SOS_Get_Analog ( Mode : Integer; Var Value, RetCode ) : Boolean;

{  Reads the analog and digital inputs form an Apple III Joystick connected
   to port A or port B. }

{  Input Values :
    Mode        : The port and the the kind of read.

                   ∅ = Port B, buttons only
                   1 = Port B, buttons and X-axis
                   2 = Port B, buttons and Y-axis
                   3 = Port B, buttons and X&Y-axis
                   4 = Port A, buttons only
                   5 = Port A, buttons and X-axis
                   6 = Port A, buttons and Y-axis
                   7 = Port A, buttons and X&Y-axis

                  All other values are INVALID.

  Output Values :

    Value       : The port value as specified in the mode passed:

                  For the buttons ∅ = False; 255 = TRUE.

                  Byte ∅ = Button ∅
                  Byte 1 = Button 1
                  Byte 2 = X-axis
                  Byte 3 = Y-axis

                  Use a packed array [∅..3] of ∅..255.

    RetCode     : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Terminate : Boolean;

{ Terminates the currently executing program and interpreter. }

{$ENDC}
```

```
{ SOS Memory Management calls--use with care...}
{$IFC SOS_Memory_Mgt}

Function SOS_Request_Seg ( Base, Limit, SegId : Integer; Var SegNumb,
                           RetCode ) : Boolean;

{  Requests a certain type and size of memory segment from SOS.}

{  Input Values :

     Base       : The segment address of the beginning of the request.
     Limit      : The segment address of the end of the request.
     SegId      : The type of segment being requested ($2Ø to $7F is user).

   Output Values :

     SegNub     : The segment number that SOS uses to keep track of segment
                  with.
     RetCode    : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Find_Seg ( SrchMode, SegId : Integer; Var FiveInts ) : Boolean;

{  Finds a memory segment of a certain type--if possible.  If not, then
   the largest free segment is returned along with a SEGRQDN ($E1) error.}

{  Input Values :

     SrchMode   : Ø = may not cross a 32K bank boundary; 1 = may not cross
                  more than one 32K bank boundary; 2 = may cross any number
                  of bank boundaries.
     SegId      : The type of segment being requested ($2Ø to $7F is user).
     FiveInts.Page : The number of pages to find for the segment (see FiveInts
                     Pascal data type).

   Output Values :

     The returned record should have the following type :

       FiveInts = Packed Record of
                   Pages   : Integer;  (* The largest number of pages *)
                   Base    : Integer;  (* The first page number *)
                   Limit   : Integer;  (* The last page number *)
                   SegNumb : Integer;  (* The SOS segment number*)
                   RetCode : Integer;  (* The SOS return code *)
                  End;

}

Function SOS_Change_Seg ( SegNumb, ChgMode : Integer; Var Pages,
                          RetCode ) : Boolean;

{  Changes either the base or limit segment address of the specified
```

```
   segment by adding or releasing the number of pages passed in the
   pages parameter.  If the request is not possible, then SEGRQDN ($E1)
   is returned and the maximum allowable page count is returned in Pages.}

{  Input Values :

     SegNumb     : The segment number of the memory segment to be changed.
     ChgMode     : Ø = Release from the base; 1 = add before the base; 2 =
                   add after the limit; 3 = release from the limit.
     Pages       : The number of pages to add or release from the segment.

   Output Values :

     Pages       : The maximum number of pages that can be added or removed.
     RetCode     : An integer to contain the SOS return code (a zero means no
                   errors).
}

Function SOS_G_Seg_Info ( SegNumb : Integer; Var FiveInts ) : Boolean;

{  Returns the beginning and ending locations, size in pages, and id code
   of the segment specified by SegNumb.}

{  Input Values :

     SegNumb     : The SOS segment number.

   Output Values :

     The returned record should have the following type :

       FiveInts = Packed Record of
                   Base    : Integer;  (* The first page number *)
                   Limit   : Integer;  (* The last page number *)
                   Pages   : Integer;  (* The number of pages *)
                   SegId   : Integer;  (* The SOS segment identifier *)
                   RetCode : Integer;  (* The SOS return code *)
                  End;

}

Function SOS_G_Seg_Numb ( SegAddr : Integer; Var SegNumb, RetCode ) : Boolean;

{  Returns the segment number of the segment, if any, that contains
   the specified segment address.}

{  Input Values :

     SegAddr     : The segment address in question.

   Output Values :

     SegNumb     : The segment number of the segment that contains the
                   specified segment address.
```

```
    RetCode     : An integer to contain the SOS return code (a zero means no
                  errors).
}

Function SOS_Rel_Seg ( SegNumb : Integer; Var RetCode ) : Boolean;

{  Release the segment with the specified segment number, if any.}

{  Input Values :

    SegNumb     : The SOS segment number.

   Output Values :

    RetCode     : An integer to contain the SOS return code (a zero means no
                  errors).
}
{$ENDC}
```

```
                    SOS Plus Calls--February 1, 1983          Version 1.1

{ Additional SOS/Pascal support routines (these are NOT calls into SOS). }

{$IFC SOS_Plus_IO}
Function Up_Load ( Var Char_Set ) : Boolean;

{  Uploads the SOS character set from $CØØ - $FFF into the 1Ø24 byte buffer
   which is supplied by the caller. }

{  Output Values :

     Char_Set : a packed array [Ø..1Ø23] of char which receives the current
                SOS character set image.
}

Function  At_Sign ( Var Object ) : Integer;

{  Returns the address of a Pascal variable as an integer.  }

{  Input Values :

     Object :  The Pascal variable to be converted.

   Returned Result :

     The passed variable as an integer value.
}

{$ENDC}
```

Implementation

```
 {$IFC SOS_Device_IO}
 Function SOS_D_Status; External;
 Function SOS_D_Control; External;
 Function SOS_Get_D_Num; External;
 Function SOS_D_Info; External;
 {$ENDC}

 {$IFC SOS_File_IO}
 Function SOS_Create; External;
 Function SOS_Destroy; External;
 Function SOS_Rename; External;
 Function SOS_Set_Info; External;
 Function SOS_Get_Info; External;
 Function SOS_Volume; External;
 Function SOS_Set_Prefix; External;
 Function SOS_Get_Prefix; External;
 Function SOS_Open; External;
 Function SOS_New_Line; External;
 Function SOS_Read; External;
 Function SOS_S_Read; External;
 Function SOS_Write; External;
 Function SOS_S_Write; External;
 Function SOS_Close; External;
 Function SOS_Flush; External;
 Function SOS_Get_B_Mark; External;
 Function SOS_Get_B_EOF; External;
 Function SOS_Set_B_Mark; External;
 Function SOS_Set_B_EOF; External;
 Function SOS_Get_Mark; External;
 Function SOS_Get_EOF; External;
 Function SOS_Set_Mark; External;
 Function SOS_Set_EOF; External;
 Function SOS_Set_Lev; External;
 Function SOS_Get_Lev; External;
 {$ENDC}

 {$IFC SOS_Utility_IO}
 Function SOS_S_Fence; External;
 Function SOS_G_Fence; External;
 Function SOS_Set_Time; External;
 Function SOS_Get_Time; External;
 Function SOS_Get_Analog; External;
 Function SOS_Terminate; External;
 {$ENDC}

 {$IFC SOS_Memory_Mgt}
 Function SOS_Request_Seg; External;
 Function SOS_Find_Seg; External;
 Function SOS_Change_Seg; External;
 Function SOS_G_Seg_Info; External;
 Function SOS_G_Seg_Numb; External;
 Function SOS_Rel_Seg; External;
 {$ENDC}

 {$IFC SOS_Plus_IO}
 Function Up_Load; External;
 Function  At_Sign; External;
 {$ENDC}


 Procedure SOS_Data; External;

End.  { Of Unit SOS }
```