PERQ Systems Corporation

C SYSTEM INTERFACES

Part One

June 30, 1985

This document is for use with C Version 2.0, which runs under Accent Version S6 with Amendment No. 2.

2600 Liberty Avenue
P. O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Table of Contents — Page

1. Introduction

This document is intended for Accent users who are programming in the C language. It explains how to interface C programs with the Accent function library and how to migrate from a Pascal programming environment to a C programming environment.

Chapter 5 contains a sample program that demonstrates how to convert from C strings to Pascal strings for Matchmaker-interfaced functions and how to invoke window manager calls.

For the user's convenience all of the header (.h) files in the C library are listed in Chapter 6.

As a general reference to the C language, see the following manuals (both are available from PERQ Systems):

- *The C Programming Language*, by Brian W. Kernighan and Dennis M. Ritchie of Bell Laboratories, Prentice-Hall, 1978 (PQS Part No. 655043-00)

- *C, A Reference Manual*, by Samuel P. Harbison and Guy L. Steele, Jr., Tartan Laboratories, 1984 (PQS Part No. 655029-01)

PERQ C conforms more closely to the description in the first manual. However, only the functions listed in this document are available, not necessarily all of the functions discussed in either of the above manuals.

2. Interfacing C Programs with the Accent Function Library

The routines in the Accent function library (found in the directory LibPascal) are available, with a few exceptions, from the C programming environment. In most cases, the use of this library from the C programming environment is similar to its use from the Pascal programming environment. However, there are some exceptions to this rule which will be explained in the following sections.

Most of the Accent server functions are currently implemented in Pascal, and programmers using these servers are expected to be aware of the implications of interfacing to a Pascal process from a process running in another language. Specifically, any functions that use a Matchmaker interface to communicate with another process or server assume this Pascal interface (see the document "Matchmaker: The Accent Remote Procedure Call Language" in the *Accent Languages Manual*). Since Matchmaker generates the interfaces, most of the low-level interfacing work is done for the programmer. However, certain types of parameters (such as strings) must be in the Pascal format for these interfaces to work as expected. Other functions that are not directly generated by the Matchmaker facilities may assume a C-like interface and generate the appropriate Pascal compatibility code. Always check the parameter declarations in the procedure headings section for the correct type of parameter to use.

Note: Eventually this will be transparent to the programmer. Matchmaker is capable of storing C and Pascal strings in a common format and unpacking the string into the correct format for each language. However, the current release of the C

environment has not taken advantage of these capabilities.

2.1. Differences Between the C and Pascal Environments

Programmers wishing to use the Accent function library from the C programming environment must be aware of some differences between the C and Pascal programming environments. These differences can be critical when interfacing to the Accent function library.

The differences, discussed in detail in the following sections, are in the areas of:

- parameter passing
- pre-defined type sizes
- arrays
- strings
- pointers
- Boolean values
- Matchmaker ServerMessage implementations

Also please note that under the current release memory obtained through malloc is permanently assigned to the process. The free routine does not make this memory available again (even for subsequent mallocs). If space is a concern, you should make calls to validate and invalidate memory as declared in AccentUser.h.

2.1.1. Parameter Passing

There are three common methods of passing parameters. Pascal uses two of these: call-by-reference and call-by-value. In call-by-reference, the calling routine passes to the subroutine the virtual address of the parameter. In call-by-value, the parameter is evaluated and the resulting value is passed to the subroutine. The use of a call-by-reference parameter may result in a change of the contents of the referenced variable. A call-by-value parameter never changes the contents of the parameter. In fact, it is common to use an arithmetic or logical expression as a call-by-value parameter. In Pascal routine declarations, any parameter that is preceded by the keyword "VAR" is declared to be a call-by-reference parameter. All others are call-by-value.

In C, all parameters are call-by-value parameters. It is possible, however, to simulate a call-by-reference parameter in C by passing the address of an item in the procedure call and then operating on that item within the procedure by dereferencing the address.

The *Accent Programming Manual* describes the procedural interfaces to each of the Accent server processes in Pascal format. Therefore, if a parameter to a particular routine is described as a VAR parameter of type XTYPE, the C version of this library function would declare a parameter of type "XTYPE *" (a pointer to a variable of type XTYPE). If a C program declared an item X of type XTYPE, the user could invoke the routine with "&X" as an argument (i.e., the address of X).

2.1.2. Pre-Defined Type Sizes

You must be careful of type definitions that appear to be similar in C and Pascal but which actually differ in size. For example, an integer in Pascal is 16 bits long. In the C language, an integer (int) is 32 bits long. In order to satisfy a Pascal server requirement, Matchmaker would define the C type to be of type short int (16 bits). However, the C programmer must make sure that a 32 bit int variable is not passed as the 16 bit short int parameter.

In C, predefined types have the following sizes:

long	32 bits
int	32 bits
short int	16 bits
short	16 bits
unsigned short	16 bits
unsigned char	8 bits
char	8 bits
<any type> *	32 bits
unsigned (long)	32 bits
float	32 bits

double	32 bits

2.1.3. Passing Arrays as Parameters

Remember that all parameters in C are call-by-value. This fact, along with the fact that in C an array name is the address of the first element of the array, means that a copy of the array is not passed to the routine; only the virtual address of the array is passed.

Note that in Pascal, a call-by-value array parameter may be passed to a subroutine (and, in fact, enough space for it would be set aside in the local stack). This would not happen in a C subroutine unless the array was "hidden" inside a structure or union. (In a structure or union, the entire item is passed--not just a pointer to it.)

2.1.4. Strings

You should consult Chapter 6 of this document to check the declaration type of a particular parameter. In the C environment, strings are not supported as a separate data type but are represented as an array of characters. The array is indexed from zero and is terminated by a null character ('\0'). Therefore the first character is stored in the first element of the array (string[0]). Each element in the array is eight (8) bits long.

In the Pascal environment, strings are supported as a separate data type. The string type is also an array of eight (8) bit entities. However, the first entity is not a character but is used to dynamically store the length of the string. The first character is stored in the second element of the string (string[1]).

A function assuming a Pascal-type compatibility would therefore expect a byte describing the length of the string to precede the

characters in the string. A C-type string passed to such a module through a Matchmaker-generated interface would result in the first character of the string being interpreted as some number x and the following x bytes interpreted as the intended string.

In order for C programs to correctly prepare and pass Pascal strings across the interface, a type-defining macro is defined in C_Types.h. This defines a Pascal-compatible string as follows:

Code:

```
#define  DefineString(NameOfType, SizeOfString)

   typedef  union {
                unsigned char    _StrSize;
                char             _StringChars[SizeOfString+1];
                } NameOfType
```

To use this, an interface would invoke

DefineString(NewStringType, NumberofCharacters);

and then declare some variable NewString as

NewStringType NewString

The length of the string would be assigned to

NewString._StrSize

The characters of the string would be assigned as

NewString._StringChars[x+1] = RealCString[x];

If NewString._StringChars[LastElement+1] is then filled in with a '\0' (which is used in the C environment to designate the end of a string), the structure would contain strings compatible with both the Pascal and the C environments. (However, the real C string would start at NewString._StringChars[1].)

Since the string resides inside a structure, and since whole structures may be passed to a subroutine, this Pascal-compatible string may be passed through a parameter list. An example of this would be:

```
Some_Procedure(... ... NewString, ... ...);
```

If, however, only a pointer to this string was expected by the procedure, the string would be passed as follows:

```
Some_Other_Procedure(... &NewString, ...);
```

Once again, check the module headings in Chapter 6 for the type of string to be passed.

The file Strhack.h contains macros that copy Pascal to C strings and vice versa. Use of these macros is documented in that file. Also see the sample program in Chapter 5, which demonstrates how to convert from C strings to Pascal strings for Matchmaker interfaced functions.

2.1.5. Pointer formats

Unlike Pascal, it is possible in the C language to address individual bytes. However, because the PERQ is a word-addressable machine (where words are 16 bits long), the format for pointers to bytes or chars (8 bits) and pointers to shorts (16 bits) or ints (32 bits) differs.

Pointers in either Pascal or C are 32 bit quantities. A short or int pointer in C uses the entire 32 bits as address, but a char pointer takes the bottom 31 bits of address, shifts them left by 1 bit, and then uses the 0 bit to signify reference to the lower or upper byte of the word.

You need only to be aware that pointers to chars and pointers to words differ. The compilation system generates the appropriate code for conversion between char and word pointers. However, you must always explicitly cast (with char *) a word pointer to a char pointer. Likewise, you must note that assignment of a char pointer to a word pointer is not possible if the char pointer points to the upper byte of a word.

Note that the malloc function always returns a character pointer to the area set aside for program use. This pointer is always word aligned. You should take care, however, to cast the pointer to the appropriate type (structure_name *, int *, etc.).

2.1.6. Boolean values

In Pascal, Boolean values are represented as true and false, but in C they are represented as integers. Zero is considered "false" and all nonzero values are considered "true" (the integer 1 is the standard "true" value).

2.1.7. Matchmaker ServerMessage implementations

One type of remote procedure call interface in a Matchmaker-generated user-server interface is a ServerMessage call. For a complete description of these, see the document "Matchmaker: The Accent Remote Procedure Call Language" in the *Accent Languages Manual*. For our purposes a ServerMessage is one which is asynchronously generated by the Server without the user side sending a request message and then waiting for a reply.

The code generated by Matchmaker for the Pascal environment (procedure XNAME) would raise an exception for this call when the message was received on the user side. If the user code did not declare a handler for this exception, the exception would eventually cause the system handler to report "uncaught exception: NAME." In any case, the user program would compile and run whether or not a handler was declared for exception NAME.

There are currently no run-time exception mechanisms in C which are equivalent to those implemented in the Pascal run-time system. The code generated by Matchmaker for the C environment (procedure XNAME) would make a call to procedure NAME (since there are no exception mechanisms), with the same parameters that would have been passed as the Pascal exception was raised. Procedure NAME, however, is passed to procedure

XNAME as a parameter. Therefore, in the C environment the user will find procedure XNAME declared with an additional parameter which is a pointer to a function. The user would then implement procedure NAME and pass a pointer to this function when procedure XNAME is invoked.

2.2. Including the Right Files

In order to have access to a particular function in the Pascal environment, the user program would import the particular file that defines the exported routines, types, and variables that will be used. In the C environment, you would include two different files. One file, usually <AnyName>Defs.h, contains #include and #define directives, type definitions, and occasionally variable definitions. The other file, <AnyName>.h, has a listing of the externally defined routines in the format:

```
extern type_it_returns routine_name();
```

This file tells the compiler that routine_name is a function that is defined in another file and that it returns a value of type type_it_returns.

As an example, if the user program were making calls to the window manager (Sapphire), it should include Sapph.h for the external declarations of the window manager functions and SapphDefs.h for type definitions and #define directives. One of the function declarations in Sapph.h is as follows:

```
extern Window CreateWindow ( );
```

This declaration tells the compiler that a function called CreateWindow has been linked into Libc.lib and that the value returned by the function is of type Window.

3. Interface to the Servers

As documented in the *Accent Programming Manual*, the Accent user may invoke the services of any of the Accent server processes. The user interfaces to these servers are available in the C environment by linking the C program with the library Libc.lib. Since Libc.lib also contains initialization code (which corresponds to Pascalinit in the Pascal environment), all C programs should be linked with Libc.lib, whether or not any of the Accent server functions are invoked.

3.1. Globally Available Items

The C header file crt0.h lists the variables which are globally available to the user through the crt0 initialization module. These are:

int ComputingEnvironment
: Indicates nature of parent process. Is 0 for Accent, >0 for QNIX.

int InPorts_Cnt
: Number of Ports passed to process in InPorts array

Port TimePort
: Connection to Time Server

Port SesPort
: Connection to File Server

Port EMPort
: Connection to Environment Manager

Port PMPort
Connection to Process Manager

Port NameServerPort
Connection to Name Server (used for calls to "lookup")

Port SapphPort
Connection to Window Manager

Port TypescriptPort
Connection to Typescript Manager

Port UserWindow
Connection to current Window

Port UserTypescript
Connection to current Typescript

Boolean UserWindowShared
Boolean describing whether the window is shared.

PortArray * InPorts
Pointer to array of Ports which is passed to the process at process initialization. This array of ports is set up by the parent process and sent to the child process via an Accent fork. If the parent of the process is the Accent or QNIX shell, this array will be empty.

These globally defined variables are useful while invoking the server fuctions. For example, when a file system function parameter is listed as "ServPort" the port SesPort would contain the "ServPort" connection to the file server.

3.2. Accessing the Server Functions

To access any of the server functions, the user need only link with Libc.lib. However, the program should include the header files for all of the servers being accessed. See Section 2.2 for a discussion of how to include the right files.

4. Adapting the Programming Environment

4.1. Finding the C Execution Environment

When a C program is linked, the C library provides access to all of the Accent library functions that have been implemented in C much the same way as LibPascal provides access to all of the Accent library functions that are implemented and available from Pascal. The standard way to link a C program is explained in the document "PERQ C Programming" in the *Accent Languages Manual*; briefly, the command is:

```
lnk -o program.exe libc.lib program.o
```

The C preprocessor (cpp), the C compiler(pcc), the assembler (asm), and the linker (lnk) are normally kept in one directory. In order to enable the loader to find the executable files when these commands are invoked, place the directory containing the C utility executable files on the run search list (run:), as described under Define in the document "User Facilities" in the *Accent User's Manual.*

4.2. Directing the Compiler to the Include Files

In the C programming language there are two standard ways to bracket a file name referenced within an " #include" statement. One method is to enclose the file within double quotes. Double quote enclosure will direct the preprocessor (cpp) to look for the file first in the current directory and then in the directories on the Default: search list.

The second method is to use angle brackets (< >) to bracket

the file. Angle brackets direct the preprocessor to look for the file in the standard library directory. However, this standard library directory is not globally defined for the user since the user has the option of placing the library in any directory that is deemed appropriate. The C preprocessor (cpp) is directed to look first in the standard library directory(ies) indicated by the "-I" (include) switch on the command line of the cpp invocation and then in the directories on the default search list. For each directory that is to be searched but which is not on the default search list, use the following switch:

-I< DirectoryName>

Hence an invocation such as

```
cpp a.c a.i -I/sys/accent/native_c/ -Iinc:
```

would look in the /sys/accent/native_c/ directory and in the directory that has been defined as inc: and then in the directories on the default search list. Note that the fully specified path name must end in a slash ("/") and that a defined environment variable may be used to specify an include directory. Note also that there are no spaces between the "-I" switch and the directory name.

4.3. Invoking the C Debugger

A primitive debugger, named CMace, is available for the Accent Native C environment (the C environment that does not use the QNIX facilities). In order to invoke this debugger, the Accent environment must be modified to either automatically invoke this debugger or to prompt for the name of the debugger to run.

If you program in the C language only, you may insert the following command into your InitialShell.Cmd file (or issue this

command to the shell):

define debuggername /<path>/cmace -global

<Path> is the path to the directory in which cmace.run is located. Note, however, that if a Pascal program falls into the debugger, CMace will not be able to operate on the Pascal run file.

If you program in both C and Pascal, you may insert the following command into your InitialShell.Cmd file (or issue this command to the shell):

define debuggername ? -global

The process manager will then prompt you in the process manager window for the name of the debugger to use when an error occurs in a user program. Make the process manager window the listener and then type the name of the debugger run file.

To obtain help after the CMace debugger has been invoked, type *? <RETURN>* for a summary of commands.

5. Sample Program

Below is a sample program that demonstrates how the user must convert from C strings to Pascal strings for Matchmaker interfaced functions (as described in Section 2.1.4). It also shows how to invoke Sapphire calls. References in the program comments to Sapphire documentation refer to the document "The Window Manager" in the *Accent Programming Manual.*

```
/*
 *  PERQ Systems Corporation
 *  Copyright (C), 1985

 */

/*  Header files needed:  */

#include <stdio.h>
#include <strhack.h>        /* Needed in conversion of C
                                and Pascal strings          */
#include <viewpt.h>         /* Window viewport routines    */
#include <sapph.h>          /* Sapphire routines           */
#include <pathname.h>       /* Function FindPathName       */
#include <crt0.h>           /* UserWindow defined          */



/*  Variable Declaration  */

short           leftx, topy, width, height, rank;
Boolean         fixedposition, fixedsize, hastitle, hasborder,
                    hasicon, memory, courteous, transparent;
GeneralReturn GR;
Window          parentwindow, childwindow;
Viewport        pvp, vp, fontvp;
```

```
TitStr          title;
ProgStr         progname;
APath_Name      fontfile;
Entry_Type      *pentrytype;
Name_Status     *pnamestatus;
char            *cprogname, *ctitle, *cfontfile, *looklist;
char            answer[80];
LineFunct       lfunct;
short           x1, y1, x2, y2, i, j;


main()
{
   /*  Set the variables to be used as parameters in
       function CreateWindow  */

   parentwindow = UserWindow;
   fixedposition = FALSE;
   leftx = 50;
   topy = 0;
   fixedsize = FALSE;
   width = 500;
   height = 500;
   hastitle = TRUE;
   hasborder = TRUE;
   hasicon = TRUE;

   /* CreateWindow uses strings of the Pascal format (see
      Section 2.1.4) and therefore the user must convert
      C strings to Pascal strings before the call to the
      Sapphire function.  The routines for converting
      strings are declared in strhack.h.
   */

   ctitle = "Newly Created Window";
   C_To_Pascal(title, ctitle);

   cprogname = "crewin";
   C_To_Pascal(progname, cprogname);
```

```
/* Function CreateWindow creates a new window on the
   screen.  The parameters are described in detail
   in the Sapphire documentation in the chapter on
   window and viewport routines.  The main
   differences between the Pascal system call and
   the C system call are 1) the conversion of string
   parameters as described above, and  2) the call-by-
   reference parameters in Pascal will be preceded by
   an '&' in the C function call (see Section 2.1.1).
*/

childwindow = CreateWindow(parentwindow,
                           fixedposition,
                           &leftx,&topy,
                           fixedsize,
                           &width,&height,
                           hastitle,hasborder,
                           title,&progname,
                           hasicon,&vp);



/* Function SetWindowAttention will display the attention
   flag in the icon of the new window.  This function is
   explained in detail in the Sapphire documentation in
   the chapter on icon routines.
*/

SetWindowAttention(childwindow, TRUE);
printf("Type 'a' to continue: ");
scanf("%s", answer);


/*  Function VPLine draws a line in the viewport 'vp.'
    This viewport was assigned in the call to CreateWindow
    (last parameter).   VPLine is explained in the
```

```
    Sapphire documentation in the chapter on graphics
    primitive routines.  An example of the graphics
    available through Sapphire can be seen if you compile
    and run this program.
*/

 lfunct = XORLine;
 x1 = 0;
 y1 = 0;
 x2 = width;
 y2 = height;

 VPLine(vp,lfunct,x1,y1,x2,y2);

 for (j=100; j >= 1; j--)
    {
       y1 = 0;
       y2 = height;
       i = -j;
       while (i < width)
             {
                x1 = i + j;
                x2 = width - x1;
                VPLine(vp,lfunct,x1,y1,x2,y2);
                i = i + j;
             }
       i = -j;
       while (i < height)
             {
                y1 = i + j;
                y2 = height - y1;
                VPLine(vp,lfunct,x1,y1,x2,y2);
                i = i + j;
             }
    }


 /* LoadFontFile and FindPathName are two examples of
```

```
        functions that are not Matchmaker defined.  Therefore
        the string parameters for these two functions are
        C strings (char *) and not Pascal strings.  Function
        FindPathName is described in the section on interface
        operations in the document "The File System" in the
        Accent Programming Manual, and the parameters for
        C can be found in the header file PathName.h.
        LoadFontFile is defined in the Sapphire documentation
        in the chapter on routines for loading data files,
        and the C declaration is in SapphLoadFile.h.

        In this example, LoadFontFile will try to read in a
        font from the disk called "windowmanager2.kst."  This
        call will not be successful because the file will not
        be found.  It is only a demonstration of how to make
        the call.  For a complete Pascal sample application
        program utilizing other Sapphire calls, please refer
        to the Sapphire documentation.
    */

    cfontfile = "windowmanager2.kst";
    looklist = "current";
    GR = FindPathName(cfontfile,looklist,TRUE, &pentrytype,
         &pnamestatus);

    fontvp = LoadFontFile(vp,cfontfile);

}
```

6. Header Files

This chapter lists all of the .h (header) files that are found in the C library. All of the routines defined in these header files are linked into libc.lib.

If the abstract, parameters, and returns are not listed for a function, they are the same as in the Pascal library. The opening paragraphs of the section containing that function will refer you to the document that contains detailed information (either one of the documents on the Accent servers in the *Accent Programming Manual* or the document "The Pascal Library" in the *Accent Languages Manual*). The indexes in the back of the *Accent Programming Manual* and the *Accent Languages Manual* will help you to find the function quickly.

All of the functions available from the C library are listed in the index that follows this document.

6.1. ACB

File ACB.h defines offsets within the Activation Control Blocks (ACB's) in the PERQ memory stack. The general form of an ACB is shown below. For a more detailed explanation, see the document "Pascal/C Machine Reference" in the *Accent Microprogramming Manual.*

+0 Static link - address of the ACB of the surrounding procedure.

+1 Local pointer - address of the current local variables.

+2 Dynamic link - address of the previous ACB on the stack.

+3 Global link - address of the previous routine's global data.

+4 Top link - address of the previous top-of-stack.

+5 Return segment - segment number of the previous procedure.

+6 Return address - offset within the return segment.

+7 Return routine number - previous routine number.

+8 Exception pointer - pointer to list of exception enable blocks.

+9 E-Stack size - number of save words of expression stack.

+10..+n+9 E-stack image - previous expression stack.

Code:

```
/* static link */
```

```
#define ACBSL    0
/* local pointer */
#define ACBLP    1
/* dynamic link */
#define ACBDL    2
/* global link */
#define ACBGL    3
/* top link */
#define ACBTL    4
/* return segment */
#define ACBRS    5
/* return address */
#define ACBRA    6
/* return routine number */
#define ACBRR    7
/* exception pointer */
#define ACBEP    8
/* E-stack size */
#define ACBStackSize    9
/* saved E-stack */
#define ACBSaveStack  10
/* ACB size + maximum E-stack */
#define ACBReserve  26
/* largest ACB */
#define ACBLength  26
```

6.2. AccCall/AccCallDefs

Files AccCall.h and AccCallDefs.h contain the exported procedure calls and the definitions for the Accent primitives that are implemented by system calls rather than by messages.

The calls are explained in the document "The Kernel Interface" in the *Accent Programming Manual.*

6.2.1. AccCall

Code:

```
#include <AccentType.h>
#include <AcccallDefs.h>

extern GeneralReturn Send();
  /*  Msg             *xxmsg;
      Long             maxwait;
      SendOption       option;
  */

extern GeneralReturn Receive();
  /*  Msg             *xxmsg;
      Long             MaxWait;
      PortOption       PortOpt;
      ReceiveOption    Option;
  */

extern GeneralReturn SetPortsWaiting();
  /*  PortBitArray *Ports;  */


extern GeneralReturn PortsWithMessages();
  /*  Long             msgType;
      PortBitArray    *Ports;
  */
```

```
extern GeneralReturn MoveWords();
  /*  VirtualAddress   SrcAddr;
      VirtualAddress  *DstAddr;
      Long             NumWords;
      Boolean          Delete;
      Boolean          Create;
      Long             Mask;
      Boolean          DontShare;
  */


extern GeneralReturn SoftEnable();
  /*  Boolean NormOrEmerg;
      Boolean EnOrDis;
  */


extern GeneralReturn GetIOSleepID();
  /*  Long    *SleepId;  */

extern GeneralReturn EReceive();
  /*  Msg             *xxmsg;
      Long             MaxWait;
      PortOption       PortOpt;
      ReceiveOption    Option;
  */


extern GeneralReturn RectRasterOp();
  /*  Port    DstRectangle;
      short Action;
      short DstX;
      short DstY;
      short Width;
      short Height;
      Port    SrcRectangle;
      short SrcX;
      short SrcY;
```

```
  */


extern GeneralReturn RectDrawLine();
  /*  Port     DstRectangle;
      short Kind;
      short X1;
      short Y1;
      short X2;
      short Y2;
  */


extern GeneralReturn RectPutString();
  /*  Port            DstRectangle;
      Port            FontRectangle;
      short           Action;
      short          *FirstX;
      short          *FirstY;
      String255      *StrPtr;
      short           FirstChar;
      short          *MaxChar;
  */


extern GeneralReturn  RectColor();
  /*  Port            Rectangle;
      short           Action;
      short           X;
      short           Y;
      short           Width;
      short           Height;
  */


extern GeneralReturn  RectScroll();
  /*  Port            Rectangle;
      short           X;
      short           Y;
```

```
        short           Width;
        short           Height;
        short           XAmt;
        short           YAmt;
    */


extern GeneralReturn LockPorts();
    /*  Boolean          LockThem;
        Port            *Ports;
        Long             PortsCount;
    */



extern GeneralReturn MessagesWaiting();
    /*  Long             MsgType;
        Port            **Ports;
        Long             *PortsCount;
    */
```

6.2.2. AccCallDefs

Code:

```
#include <Accenttype.h>

#define MAXLOGMESS 20
#define MAXMSGDATA 20
#define LOGMSGS  0

typedef struct {
                    Msg  H;
                    short D[MAXMSGDATA];
                }   LMsg;

typedef LMsg * pLMsg;

typedef struct{
```

```
        short Init;
        long  MsgsSent;
        long  MsgsRec;
        short NxtMsg;
        struct {
                short Sent;
                short InProg;
                GeneralReturn GR;
                LMsg M;
               }    LMsgs[MAXLOGMESS + 1];
    }   MsgLog;

typedef MsgLog * pMsgLog;
```

6.3. AccentType

File AccentType.h contains the types used by the Accent operating system and user interfaces to Accent.

See also the document "Pascal Library" in the *Accent Languages Manual.*

Code:

```
#include <C_Types.h>

/*
 *    Constant:
 *      — PAGEBYTESIZE, PAGEWORDSIZE, PAGEBITS, DISKBUFSIZE
 *
 *    Purpose:
 *        Constants which define the size of an Accent physical and
 *        disk page.
 */


#define PageByteSize  512
                      /* Number of bytes in physical page.*/
#define PageWordSize  (PageByteSize / 2)
                      /* Number of 16 bit words per page. */
#define DiskBufSize   (PageByteSize / 2)
                      /* Number of 16 bit words per disk page.*/
#define PageBits      8
                      /* Number of bits needed to represent
                         a page. */


typedef         unsigned char   Bit8;
typedef         unsigned short  Bit16;
typedef         int32           Bit32;

#define NBITSINWORD 16
```

```
#define _X_AnyPtr(x) ((pointer) x)
#define _AnyPtr(x) ((pointer) x)
#define _X_Lng(x) ((long) x)
#define _Lng(x) ((long) x)
#define _Bit32Ptr(x) ((long *) x)
#define _X_Bit32Ptr(x) ((long *) x)
#define _Word0(x) (x)
#define _Word1(x) (x >> NBITSINWORD)
#define _X_Word0(x) (x & ~(~0 << NBITSINWORD))
#define _X_Word1(x) (x & (~0 << NBITSINWORD))
#define _Byte0(x) (x)
#define _Byte1(x) (x >> 8)
#define _Byte2(x) (x >> 16)
#define _Byte3(x) (x >> 24)
#define _X_Byte0(x) (x & ~(~0 << 8))
#define _X_Byte1(x) (x & (~(~0 << 8) << 8))
#define _X_Byte2(x) (x & (~(~0 << 8) << 16))
#define _X_Byte3(x) (x & (~(~0 << 8) << 24))
#define _X_PageOffset(x) (_X_Byte0(x))
#define _X_LswPage(x) (_X_Byte1(x))
#define _X_MswPage(x) (_X_Word1(x))
#define _PageOffset(x) (_Byte0(x))
#define _LswPage(x) (_Byte1(x))
#define _MswPage(x) (_Word1(x))
#define _bPageOffset 16
#define _sPageOffset 0
#define _X_Blk(x) (_X_Word0(x))
#define _Blk(x) (_Word0(x))
#define _X_Field2(x) (_X_Byte1(x))
#define _X_Field3(x) (_X_Byte2(x))
#define _X_Field4(x) (_X_Byte3(x))
#define _Field2(x) (_Byte1(x))
#define _Field3(x) (_Byte2(x))
#define _Field4(x) (_Byte3(x))

/*
  Actually bit32 is the following Pascal variant record.
  Some of the necessary macros have been defined, but
```

```
  more should be defined later.

            case integer of
             @1: ( DblWord : array [0..1] of integer);
             @3: ( Bit32Ptr: pBit32);
             @4: ( AnyPtr  : pointer);
             @5: ( Byte    : packed array [0..3] of Bit8);
             @6: ( PageOffset : Bit8;
             @     LswPage : Bit8;
             @     MswPage : Bit16
                 );
             @7: ( Lng     : Long);
             @8: ( Blk     : integer;
                   Index   : Bit12;
                   Imag    : Bit4
                 );
            @13: ( Field4  : Bit8;
            @      Field3  : Bit8;
            @      Field2  : Bit8;
                   Field1  : Bit7;
                   Field0  : Bit1);
            @14: ( Word0   : Bit16;
            @      Word1   : Bit16);
            @15: ( Byte0   : Bit8;
            @      Byte1   : Bit8;
            @      Byte2   : Bit8;
            @      Byte3   : Bit8 );

@ These have already been done (above)

*/

typedef         Bit32           *pBit32;

typedef         struct
                        {
                        Long            lsw;
                        Long            msw;
                        }
```

```
                                Bit64;

typedef         long            SegID;

typedef         short           SpiceSegKind;
#define         Temporary       0
#define         Permanent       1
#define         Bad              2
#define         SegPhysical     3
#define         Imaginary       4
#define         Shadow          5

typedef         long            VirtualAddress;
typedef         long            PhysicalAddress;
typedef         long            MicroSeconds;

typedef         union
                        {
                        long            lng;
                        char            byte[4];
                        }
                                DiskAddr;

#define EIO 0
#define CIO 1
#define FlopDrives 2
#define MultiBus 3
#define Enet 4

typedef short DiskInterface;


typedef  union {  short Unit;
                  short EAddr[3];
                }       InterfaceInfo;

/*
 *   Enumerated type:
```

```
 *        TrapCodes
 *
 *    Purpose:
 *        System trap codes.
 *
 */

typedef          short                  TrapType;

#define          TrapInit                0
#define          TrapReadFault           1
#define          TrapWriteFault          2
#define          TrapSend                3
#define          TrapReceive             4
#define          TrapSetPortsWaiting     5
#define          TrapPortsWithMessages   6
#define          TrapDebugWrite          7
#define          TrapException           8
#define          TrapNothing             9
#define          TrapRectDrawLine        10
#define          TrapRectRasterOp        11
#define          TrapCharRead            12
#define          TrapFull                13
#define          TrapFlush               14
#define          TrapMoveWords           15
#define          TrapRectPutString       16
#define          TrapError               17
#define          TrapClockEnable         18
#define          TrapGPRead              19
#define          TrapGPWrite             20
#define          TrapSoftEnable          21
#define          TrapGetIOSleepID        22
#define          TrapRectColor           23
#define          TrapRectScroll          24
#define          TrapLockPorts           25
#define          TrapMessagesWaiting     26

/*
 *        GeneralReturn
```

```
 *
 *   Purpose:
 *       Values returned from system calls and system messages.
 *
 */

typedef         short                   GeneralReturn;

#define         AccErr                    100

#define         Dummy                    (AccErr+0)
#define         Success                  (AccErr+1)
#define         TimeOut                  (AccErr+2)
#define         PortFull                 (AccErr+3)
#define         WillReply                (AccErr+4)
#define         TooManyReplies           (AccErr+5)
#define         MemFault                 (AccErr+6)
#define         NotAPort                 (AccErr+7)
#define         BadRights                (AccErr+8)
#define         NoMorePorts              (AccErr+9)
#define         IllegalBacklog           (AccErr+10)
#define         NetFail                  (AccErr+11)
#define         Intr                     (AccErr+12)
#define         Other                    (AccErr+13)
#define         NotPortReceiver          (AccErr+14)
#define         UnrecognizedMsgType      (AccErr+15)
#define         NotEnoughRoom            (AccErr+16)
#define         NotAnIPCCall             (AccErr+17)
#define         BadMsgType               (AccErr+18)
#define         BadIPCName               (AccErr+19)
#define         MsgTooBig                (AccErr+20)
#define         NotYourChild             (AccErr+21)
#define         BadMsg                   (AccErr+22)
#define         OutOfIPCSpace            (AccErr+23)
#define         Failure                  (AccErr+24)
#define         MapFull                  (AccErr+25)
#define         WriteFault               (AccErr+26)
#define         BadKernelMsg             (AccErr+27)
#define         NotCurrentProcess        (AccErr+28)
```

```
#define         CantFork                (AccErr+29)
#define         BadPriority             (AccErr+30)
#define         BadTrap                 (AccErr+31)
#define         DiskErr                 (AccErr+32)
#define         BadSegType              (AccErr+33)
#define         BadSegment              (AccErr+34)
#define         IsParent                (AccErr+35)
#define         IsChild                 (AccErr+36)
#define         NoAvailablePages        (AccErr+37)
#define         FiveDeep                (AccErr+38)
#define         BadVPTable              (AccErr+39)
#define         VPExclusionFailure      (AccErr+40)
#define         MicroFailure            (AccErr+41)
#define         EStackTooDeep           (AccErr+42)
#define         MsgInterrupt            (AccErr+43)
#define         UncaughtException       (AccErr+44)
#define         BreakPointTrap          (AccErr+45)
#define         ASTInconsistency        (AccErr+46)
#define         InactiveSegment         (AccErr+47)
#define         SegmentAlreadyExists    (AccErr+48)
#define         OutOfImagSegments       (AccErr+49)
#define         NotASystemAddress       (AccErr+50)
#define         NotAUserAddress         (AccErr+51)
#define         BadCreateMask           (AccErr+52)
#define         BadRectangle            (AccErr+53)
#define         OutOfRectangleBounds    (AccErr+54)
#define         IllegalScanWidth        (AccErr+55)
#define         CoveredRectangle        (AccErr+56)
#define         BusyRectangle           (AccErr+57)
#define         NotAFont                (AccErr+58)
#define         PartitionFull           (AccErr+59)

/*
 *   General error codes to be used
 *   by all modules that pass messages.
 */

#define         BadMsgID                1
#define         WrongArgs               2
```

```
#define         BadReply            3
#define         NoReply             4
#define         UnspecException     5
/* Message is an exception on behalf of a server */

/*
 *   Constant:
 *       MaxPorts,DEFAULTBACKLOG, MAXBACKLOG
 *
 *   Purpose:
 *       Maximum number of ports per process
 *       and number of messages held per port.
 */

#define         MaxPorts            256
#define         DefaultBacklog      0
#define         MaxBacklog          63

typedef         short       BackLogValue;   /* 0..MaxBackLog */

/*
 *   Constants:
 *       NORMALMSG,EMERGENCYMSG
 *
 *   Purpose:
 *       Possible values of MsgType field in a message header.
 *
 */

#define         NormalMsg           0
#define         EmergencyMsg        1
#define         NumMsgTypes         2

/*
 *   Constants:
 *       WAIT,DONTWAIT,REPLY
 *
 *   Purpose:
 *       Possible sending options.
```

```
 *
 */

#define         Wait                    0
#define         DontWait                1
#define         Reply                   2

typedef         short           SendOption;     /* WAIT..REPLY */

/*
 *   Constants:
 *       PREVIEW,RECEIVEIT,RECEIVEWAIT
 *
 *   Purpose:
 *       Possible sending options.
 *
 */

#define         Preview                 0
#define         ReceiveIt               1
#define         ReceiveWait             2

typedef         short           ReceiveOption;
                                        /* Preview..ReceiveWait */

/*
 *   Constants:
 *       DEFAULTPTS,ALLPTS,LOCALPTS
 *   Purpose:
 *       Possible port options on receive
 */

#define         DefaultPts              0
#define         AllPts                  1
#define         LocalPt                 2

typedef         short           PortOption;
                                        /* DefaultPts..LocalPt */
```

```
/*
 *   Constants:
 *        NULLPORT,KERNELPORT,DATAPORT,FIRSTNONRESERVEDPORT,
 *        ALLPORTS
 *
 *   Purpose:
 *        Distinguished local port numbers (or in the case
 *      of ALLPORTS a number which implies all ports).
 *
 */

#define          NullPort                0
#define          KernelPort              1
#define          DataPort                2
#define          FirstNonReservedPort  3
#define          AllPorts                (-1)

/*
 *   Constants:
 *        EXPLICTDEALLOC,PROCESSDEATH,NETWORKTROUBLE
 *
 *   Purpose:
 *        Reason for a port to be deallocated and/or destroyed.
 */

#define          ExplicitDealloc        0
#define          ProcessDeath           1
#define          NetworkTrouble         2

                                 /* ExplicitDealloc ..
                                    NetworkTrouble */
typedef          short          PortDeath;

/*   Constants:
 *        READONLY,READWRITE
 *
 *   Purpose:
 *        Protection types for virtual memory.
 *
```

```
 */

#define          ReadOnly              0
#define          ReadWrite             1

                                      /* ReadOnly .. ReadWrite */
typedef          short          MemProtection;

/*
 *   Constants:
 *        LINEARSTRUCTURE,TYPEINTEGER,TYPEPTOWNERSHIP,
 *        TYPEPTRECEIVERIGHTS,TYPEPTALLRIGHTS,
 *        TYPEPT,TYPEUNSTRUCTURED
 *
 *   Purpose:
 *        Types of message descriptors.
 *
 */

#define          TypeUnstructured      0
#define          TypeBit                0
#define          TypeBoolean            0
#define          TypeInt16              1
#define          TypeInt32              2
#define          TypePtOwnership        3
#define          TypePtReceive          4
#define          TypePtAll              5
#define          TypePt                 6
#define          TypeChar               8
#define          TypeInt8               9
#define          TypeByte               9
#define          TypeReal               10
#define          TypePStat              11
#define          TypeString             12
#define          TypeSegID              13
#define          TypePage               14

/*
 *   Constants:
```

```
 *        PORTDELETED,MSGACCEPTED,OWNERSHIPRIGHTS,
 *        RECEIVERIGHTS,INTERPOSEDONE,KERNELMSGERROR
 *        GENERALKERNELREPLY
 *
 *    Purpose:
 *        Kernel generated messages ids.
 */

#define          M_PortDeleted              ( 0100 + 001 )
#define          M_MsgAccepted              ( 0100 + 002 )
#define          M_OwnershipRights          ( 0100 + 003 )
#define          M_ReceiveRights            ( 0100 + 004 )
#define          M_GeneralKernelReply       ( 0100 + 006 )
#define          M_KernelMsgError           ( 0100 + 007 )
#define          M_ParentForkReply          ( 0100 + 010 )
#define          M_ChildForkReply           ( 0100 + 011 )
#define          M_DebugMsg                 ( 0100 + 012 )

/*
 *    Structure:
 *        Msg,ptrMsg
 *
 *    Purpose:
 *        Defines format of message header in user area.
 *
 */

/*
 *    Structure:
 *        Port
 *
 *    Purpose:
 *        Port    is the basic data structure describing a port.
 */

#if              BIT_STRUCTURES
typedef          struct
                         {
                         int                TypeName : 8;
```

```
                        int             TypeSizeInBits : 8;
                        int             NumObjects : 12;
                        unsigned
                        int             InLine : 1;
                        unsigned
                        int             LongForm : 1;
                        unsigned
                        int             Deallocate : 1;
                        }
                                TypeType;
#else
typedef         long            TypeType;


/*
 *      If not using bit fields, we can use hand-shifting.
 *      Definitions are as follows:
 *              _Foo            Function to insert into Foo
 *              _X_Foo          Function to extract from Foo
 *              _bFoo           Number of bits to shift Foo
 *              _sFoo           Size of Foo
 */

#define         _sTypeName      8
#define         _sTypeSizeInBits 8
#define         _sNumObjects    12
#define         _sInLine        1
#define         _sLongForm      1
#define         _sDeallocate    1

#define         _bTypeName      0
#define         _bTypeSizeInBits 8
#define         _bNumObjects    16
#define         _bInLine        28
#define         _bLongForm      29
#define         _bDeallocate    30

#define _TypeName(x)
          ((x&(~(-1<< _sTypeName)))      << _bTypeName)
```

```
#define _TypeSizeInBits(x)
        ((x&(~(-1<< _sTypeSizeInBits)))<< _bTypeSizeInBits)

#define _NumObjects(x)
        ((x&(~(-1<< _sNumObjects)))     << _bNumObjects)

#define _InLine(x)
        ((x&(~(-1<< _sInLine)))         << _bInLine)

#define _LongForm(x)
        ((x&(~(-1<< _sLongForm)))       << _bLongForm)

#define _Deallocate(x)
        ((x&(~(-1<< _sDeallocate)))     << _bDeallocate)

#define _X_TypeName(x)
        ((x>> _bTypeName)       &(~(-1<< _sTypeName)))

#define _X_TypeSizeInBits(x)
        ((x>> _bTypeSizeInBits)&(~(-1<< _sTypeSizeInBits)))

#define _X_NumObjects(x)
        ((x>> _bNumObjects)     &(~(-1<< _sNumObjects)))

#define _X_InLine(x)
        ((x>> _bInLine)         &(~(-1<< _sInLine)))

#define _X_LongForm(x)
        ((x>> _bLongForm)       &(~(-1<< _sLongForm)))

#define _X_Deallocate(x)
        ((x>> _bDeallocate)     &(~(-1<< _sDeallocate)))

#endif          BIT_STRUCTURES
```

```
typedef         long            Port;
typedef         Port            *ptrPort;

typedef         struct
                        {
                        Boolean         SimpleMsg;
                        long            MsgSize;
                        long            MsgType;
                        Port            LocalPort;
                        Port            RemotePort;
                        long            ID;
                        }       Msg;

typedef         Msg             *ptrMsg;

/*
 *   Types:
 *       PortArray,PortBitArray,ptrPortArray,ptrPortBitArray
 *
 *   Purpose:
 *       Used to handle arrays of ports or port conditions.
 */

typedef         Boolean         PortBitArray[MaxPorts];
typedef         PortBitArray    *ptrPortBitArray;

typedef         Port            PortArray[MaxPorts];
typedef         Port            *ptrPortArray;
typedef         Port            *ptrAllPortArray;

typedef Port LPortArray[077777];
typedef LPortArray *ptrLPortArray;

typedef Bit8 GPBuffer[8]; /* Note: The Pascal Version indexes  */
                          /* this array from 1..8              */

#define DirIOInit 0
#define DirIORead 1
#define DirIOWrite 2
```

```
#define DirIOReadCheck 3
#define DirIOWriteCheck 4
#define DirIOTrackRead 5
#define DirIOTrackWrite 6
#define DirIOParamRead 7
#define DirIOBootRead 8
#define DirIOBootWrite 9
#define DirIOClose 10

typedef short DirIOCommands;


typedef struct {  short IOStatus;
                  short UnitNumber;
                  int   PhysAddress;
                  DirIOCommands Command;
                }     DirectIOArgs;

typedef DirectIOArgs *ptrDirIOArgs;

#define DUnused 0
#define D5Inch 1
#define D14Inch 2
#define D8Inch 3
#define DSMD 4
#define DFloppy 5

typedef short DiskType;

typedef struct {
                short DskBootSize;
                short DskSectors;
                short DskNumHeads;
                short DskNumCylinders;
                int DskSecCyl;
                short HDiskType;
                union {
                        struct { short WriteCompCyl;
                                 short LandingZone;
```

```
                                    }    D5InchRec;
                            Boolean  c24MByte;
                           }caseHDiskType;
                    }    DiskParams;

typedef Bit8 DiskBuffer[DiskBufSize*2];
typedef Bit8 Header[16];

typedef struct {
        GeneralReturn ReturnValue;
        union {
                struct {
                        ptrMsg          Msg;
                        long            MaxWait;
                        short           Option;
                        short           PtOption;
                       }SendRcv;
                struct {
                        ptrPortBitArray Ports;
                        long            MsgType;
                        }Obs;
                struct {
                        VirtualAddress SrcAddr;
                        VirtualAddress DstAddr;
                        long            NumWords;
                        Boolean         Delete;
                        Boolean         Create;
                        long            Mask;
                        Boolean         DontShare;
                       }MoveWords;
                struct {
                        short           NumCmds;
                        GPBuffer        GPIBBuffer;
                       }GPIB_RW;
                struct {
                        Boolean         NormalOrEmergency;
                        Boolean         EnableOrDisable;
                       }SoftEnable;
                struct {
```

```
            long            SleepID;
            }GetIOSleepID;
struct {
            Port            RectPort;
            short           X1;
            short           Y1;
            short           X2;
            short           Y2;
            short           Kind;
           }RectLine;
struct {
            Port            SrcRect;
            Port            DstRect;
            short           Action;
            short           Height;
            short           Width;
            short           SrcX;
            short           SrcY;
            short           DstX;
            short           DstY;
           }RectRastOp;
struct {
            Port            Rect;
            Port            FontRect;
            short           Funct;
            short           FirstX;
            short           FirstY;
            short           MaxX;
            short           FirstChar;
            short           MaxChar;
            caddr_t         StrPtr;
            short           Rslt;
           }RectDrawByte;
struct {
            Boolean         LockDoLock;
            ptrLPortArray   LockPortPtr;
            long            LockPortCnt;
           }LockPorts;
struct {
```

```
                        long            MsgWType;
                        ptrLPortArray   MsgWPortPtr;
                        long            MsgWPortCnt;
                       }MsgsWaiting;
                } Ktrap;
        } Arguments;

typedef Arguments *  ptrArguments;


/*************************************************************/
/*        Process management constants and types             */
/*************************************************************/

/*
 * NB:  The following constants are carefully arranged so that
 *      the PCBHandle has all that the microcode needs to access.
 *      If you make changes here, you must make
 *      corresponding changes in the PCBHandle definition
 *      and in the microcode kernel process manager.
 */

#define         MAXPROCS        ( 63 )     /* should be power of
                                              two minus one */
#define         NUMPRIORITIES ( 16 )
#define         NUMSLEEPQS      ( 32 )     /* must be power of
                                              two */
#define         NUMQUEUES       ( NUMSLEEPQS + NUMPRIORITIES + 5 )



typedef         short           ProcState;
#define         Supervisor        0
                        /* 00 - supervisor with privileges */

#define         Privileged        1
                        /* 01 - user with privileges */

#define         BadSupervisor   2
```

```
                        /* 10 - supervisor without privileges */

#define         User            3
                        /* 11 - user without privileges */

typedef         short           ProcID;

typedef         short           PriorID;  /* 0..NumPriorities-1 */
typedef         short           QID;      /* 0..NumQueues */

typedef         short           *ptrInteger;
typedef         Boolean           *ptrBoolean;

typedef         struct
                        {
                        ProcState       State;
                        PriorID         Priority;
                        Boolean         MsgPending;
                        Boolean         EMsgPending;
                        Boolean         MsgEnable;
                        Boolean         EMsgEnable;
                        Boolean         LimitSet;
                        Boolean         SVStkInCore;
                        QID             QueueID;
                        ptrInteger      SleepID;
                        long            RunTime;
                        long            LimitTime;
                        }
                                PStatus;


/*****************************************************************/
/*          Device management constants and types                */
/*****************************************************************/

                /* maximum length for a partition name */
#define         MAXPARTCHARS        8

                /* maximum length for dev:part name */
```

```
#define         MAXDPCHARS          25

                /* maximum partitions on one device */
#define         MAXPARTITIONS       10

                /* maximum number of devices */
#define         MAXDEVICES          5

DefineString    (_PartStringType, MAXPARTCHARS);
typedef          _PartStringType  PartString;
DefineString    (_DevPartType,    MAXDPCHARS);
typedef          _DevPartType     DevPartString;

typedef         short           PartitionType;
#define         Root              0
#define         UnUsed            1
#define         Segment           2
#define         PLX               3

typedef         struct          /*entry in the PartTable*/
  {
    DiskAddr        PartHeadFree;
                        /* pointer to Head of Free List */

    DiskAddr        PartTailFree;
                        /* pointer to tail of Free List */

    DiskAddr        PartInfoBlk;
                        /* pointer to PartInfoBlock */

    SegID           PartRootDir;
                        /* SegID of Root Directory */

    short           PartNumOps;
                        /* operations done since last
                           update of PartInfoBlock */

    Long            PartNumFree;
```

```
                        /* HINT of how many free pages */

    Boolean         PartInUse;
                        /* this entry in PartTable is valid */

    Boolean         PartMounted;
                        /* this partition is mounted */

    short           PartDevice;
                        /* which disk this partition is in */

    DiskAddr        PartStart;
                       /* Disk Address of 1st page */

    DiskAddr        PartEnd;
                       /* Disk Address of last page */

    PartitionType   PartKind;
                       /* Root or Leaf */

    PartString      PartName;
                       /* name of this partition */

    Boolean         PartExUse;
                       /* Opened exclusively */

    Long            Unused;
                       /* Port is not returned */

    Boolean         PartDiskRel;
  }
                                PartInfo;

typedef         PartInfo        PartList[MAXPARTITIONS];
typedef         PartList        *ptrPartList;
```

6.4. AccentUser

File AccentUser.h contains the external declarations for routines that interface with the kernel. These routines correspond to those in the Pascal module AccInt in AccentUser.pas. They are explained in the document "The Kernel Interface" in the *Accent Programming Manual.*

Code:

```
#include <AccentType.h>

 extern void InitAccInt () ;
  /*     Port RPort;     */

 extern GeneralReturn SetBackLog ( );
  /*     Port ServPort;
         Port LocalPort;
         short BackLog;
   */

 extern GeneralReturn AllocatePort ( );
  /*     Port ServPort;
         Port  * LocalPort;
         short BackLog;
   */

 extern GeneralReturn DeallocatePort ( );
  /*     Port ServPort;
         Port LocalPort;
         Long Reason;
   */

 extern GeneralReturn IndexInterpose ( );
  /*     Port ServPort;
         Port MyPort;
         Long HisIndex;
         Port  * HisPort;
```

```
  */

extern GeneralReturn PortInterpose ( );
 /*     Port ServPort;
        Port MyPort;
        Port HisPort;
        Port  * MyNewPort;
  */

extern GeneralReturn Fork ( );
 /*     Port ServPort;
        Port  * HisKernelPort;
        Port  * HisDataPort;
        ptrPortArray  * Ports;
        Long  * Ports_Cnt;
  */

extern GeneralReturn Status ( );
 /*     Port ServPort;
        PStatus  * NStats;
  */

extern GeneralReturn Terminate ( );
 /*     Port ServPort;
        Long Reason;
  */

extern GeneralReturn SetPriority ( );
 /*     Port ServPort;
        PriorID Priority;
  */

extern GeneralReturn SetLimit ( );
 /*     Port ServPort;
        Port ReplyPort;
        Long Limit;
  */

extern GeneralReturn Suspend ( );
```

```
 /*     Port ServPort;     */

extern GeneralReturn Resume ( );
 /*     Port ServPort;     */

extern GeneralReturn Examine ( );
 /*     Port ServPort;
        Boolean RegOrStack;
        short Index;
        short  * Value;
  */

extern GeneralReturn Deposit ( );
 /*     Port ServPort;
        Boolean RegOrStack;
        short Index;
        short Value;
  */

extern GeneralReturn SoftInterrupt ( );
 /*     Port ServPort;
        Boolean NormOrEmerg;
        Boolean  * EnOrDisable;
  */

extern GeneralReturn CreateSegment ( );
 /*     Port ServPort;
        Port ImagSegPort;
        SpiceSegKind SegmentKind;
        short InitialSize;
        short MaxSize;
        Boolean Stable;
        SegID  * segment;
  */

extern GeneralReturn TruncateSegment ( );
 /*     Port ServPort;
        SegID segment;
        short NewSize;
```

```
  */

extern GeneralReturn DestroySegment ( );
 /*     Port ServPort;
        SegID segment;
  */

extern GeneralReturn ReadSegment ( );
 /*     Port ServPort;
        SegID segment;
        short Offset;
        short NumPages;
        Long  * Data;
        Long  * Data_Cnt;
  */

extern GeneralReturn WriteSegment ( );
 /*     Port ServPort;
        SegID segment;
        short Offset;
        Long Data;
        Long Data_Cnt;
  */

extern GeneralReturn ValidateMemory ( );
 /*     Port ServPort;
        VirtualAddress  * Address;
        Long NumBytes;
        Long CreateMask;
  */

extern GeneralReturn InvalidateMemory ( );
 /*     Port ServPort;
        VirtualAddress Address;
        Long NumBytes;
  */

extern GeneralReturn SetProtection ( );
 /*     Port ServPort;
```

```
        VirtualAddress Address;
        Long NumBytes;
        short Protection;
  */

extern GeneralReturn ReadProcessMemory ( );
 /*     Port ServPort;
        VirtualAddress Address;
        Long NumBytes;
        Long  * Data;
        Long  * Data_Cnt;
  */

extern GeneralReturn WriteProcessMemory ( );
 /*     Port ServPort;
        VirtualAddress Address;
        Long NumBytes;
        Long Data;
        Long Data_Cnt;
  */

extern GeneralReturn GetDiskPartitions ( );
 /*     Port ServPort;
        DiskInterface Interface;
        InterfaceInfo log_unit;
        short * unitnum;
        DevPartString * DevName;
        Long  * PartL;
        Long  * PartL_Cnt;
  */

extern GeneralReturn PartMount ( );
 /*     Port ServPort;
        DevPartString PartName;
        Boolean ExUse;
        SegID  * RootId;
        PartitionType  * PartKind;
        Port  * PartPort;
        DiskAddr  * PartS;
```

```
        DiskAddr  * PartE;
  */

extern GeneralReturn PartDisMount ( );
 /*     Port ServPort;     */

extern GeneralReturn SetTempSegPartition ( );
 /*     Port ServPort;
        DevPartString PartName;
  */

extern GeneralReturn SetDebugPort ( );
 /*     Port ServPort;
        Port DebugPort;
  */

extern GeneralReturn Touch ( );
 /*     Port ServPort;
        VirtualAddress Address;
  */

extern GeneralReturn GetPortIndexStatus ( );
 /*     Port ServPort;
        Long PortIndex;
        short  * Backlog;
        short  * NWaitingMsgs;
        short  * EWaitingMsgs;
        Port  * PortRight;
        short  * PortType;
  */

extern GeneralReturn GetPortStatus ( );
 /*     Port ServPort;
        Port PortRight;
        short  * Backlog;
        short  * NWaitingMsgs;
        short  * EWaitingMsgs;
        Long  * PortIndex;
        short  * PortType;
```

```
  */

extern GeneralReturn ExtractAllRights ( );
 /*     Port ServPort;
        Long PortIndex;
        Port  * PortRight;
        short  * PortType;
  */

extern GeneralReturn InsertAllRights ( );
 /*     Port ServPort;
        Long PortIndex;
        Port PortRight;
        short PortType;
  */

extern GeneralReturn CreateProcess ( );
 /*     Port ServPort;
        Port  * HisKernelPort;
        Port  * HisDataPort;
  */

extern GeneralReturn InterceptSegmentCalls ( );
 /*     Port ServPort;
        ptrAllPortArray  * OldSysPorts;
        Long  * OldSysPorts_Cnt;
        ptrPortArray  * SysPorts;
        Long  * SysPorts_Cnt;
  */

extern GeneralReturn DirectIO ( );
 /*     Port              ServPort;
        DirectIOArgs   * CmdBlk;
        Header          * DataHdr;
        DiskBuffer     * Data;
 */

extern GeneralReturn SetPagingSegment ( );
 /*     Port ServPort;
```

```
        SegID segment;
  */

extern GeneralReturn CreateRectangle ( );
 /*     Port ServPort;
        Port RectPort;
        VirtualAddress BaseAddr;
        short ScanWidth;
        short BaseX;
        short BaseY;
        short MaxX;
        short MaxY;
        Boolean IsFont;
  */

extern GeneralReturn DestroyRectangle ( );
 /*     Port ServPort;
        Port RectPort;
  */

extern GeneralReturn AvailableVM ( );
 /*     Port ServPort;
        Long  * NumBytes;
  */

extern GeneralReturn EnableRectangles ( );
 /*     Port ServPort;
        ptrPortArray RectList;
        Long RectList_Cnt;
        Boolean Enable;
  */

extern GeneralReturn SetKernelWindow ( );
 /*     Port ServPort;
        short LeftX;
        short TopY;
        short Width;
        short Height;
        Boolean Inverted;
```

```
    */

extern GeneralReturn Accent_Version();
  /*     Port ServPort;
         DevPartString * AccVersion;
    */

extern GeneralReturn GetRectangleParms();
  /*     Port ServPort;
         Port RectPort;
         VirtualAddress * BaseAddr;
         short * ScanWidth;
         short * BaseX;
         short * BaseY;
         short * MaxX;
         short * MaxY;
         Boolean * IsFont;
    */

extern GeneralReturn GetPermSegPort();
  /*     Port ServPort;
         Port *PermSegmentPort;
    */

extern GeneralReturn SyncDisk();
  /*     Port ServPort;
         Boolean on;
    */
```

6.5. ALoad

File ALoad.h provides facilities that are used to load and execute programs. This file is used by the shell and is not of general use to most users.

Code:

```
#include <accenttype.h>
#include <rundefs.h>
#include <pathname.h>

extern GeneralReturn ARunLoad;
   /* RunFileName,p,FileSize,
      HisKPort,LoadDebug
   */

extern GeneralReturn ShowRun;
   /* p,MapFileName    */

extern GeneralReturn ReadRun;
   /* NM,p,FileSize */

extern char *LinkTypeStr;
   /* typ */

extern char *DateString;
   /* date */

extern char LoadErrMes[256];
   /* Fatal Error message */

extern GeneralReturn LoadErrGR ();
   /* Fatal Error return code */
```

6.6. Atof

File Atof.h converts an ascii string to a floating point number.

Code:

```
#include <c_types.h>

#define LIMIT 38

extern double atof();
  /*  char * s  */
```

Parameters:

s The string of ascii characters to be converted

Returns:

The floating point number of type double (32 bits)

6.7. AuthDefs

File AuthDefs.h provides the client interface to the Authentication Server.

See also the document "Pascal Library" in the *Accent Languages Manual.*

Code:

```
#include <stdio.h>
#include <c_types.h>
#include <SesameDefs.h>

#define Auth_Var_Size 30

#define No_User 0            /* files owned by "nobody" */
#define First_User 1         /* first valid user */
#define Max_Users 1023

typedef String Auth_Var[Auth_Var_Size];

typedef short User_ID;       /* must be "bit10" */

typedef long PassType;       /* a two word value */
                               /* 4 chars for a password */

typedef struct
     {
        Auth_Var Name;           /* Name of the user */
        User_ID UserID;          /* The user ID of the user. */
        PassType EncryptPass;    /* The encrypted password. */
        APath_Name Profile;      /* Path name of the profile file. */
        APath_Name NameOfShell; /* Name of the Shell.RUN File. */
     } UserRecord;

typedef String Machine_Name[255];
```

```
#define Check_Login 0              /* user is logging in */
#define Check_User 1               /* user is changing parameters */
typedef char Check_Type;

typedef struct
     {
        User_ID UserID;
        Auth_Var UserName;
        Auth_Var MachineName;
     } Logged_User;                /* one logged-in user */

typedef int *Logged_User_List;
                  /* pointer to list of Logged_User's */

#define Auth_Error_Base 5000

#define UserNameNotFound Auth_Error_Base + 1
#define PassWordIncorrect Auth_Error_Base + 2
#define AuthPortIncorrect Auth_Error_Base + 3

#endif _authdefs
```

6.8. BootInfo

File BootInfo.h contains the definition of Boot Information Block for LibC.

Code:

```
#include <stdio.h>
#include <AccentType.h>

#define BootBlockLocation 020000000000

typedef short ASTRecord;

struct MachineInfoRec
     {
        union
            {
               short intname;
               /* configuration comes in a word */
               struct {
                        /* 0 ->  4K WCS */
                        unsigned int WCSSize:4;

                        /* 1 -> 16K WCS */
                        unsigned int Reserved:2;

                        /* True  -> Portrait screen */
                        /* False -> Landscape screen */
                        unsigned int IsPortrait:1;


                        /* IO Board revision / disk type:
                         * 0  -> CIO board with Shugart disk
                         * 1  -> CIO board with Micropolis disk
                         * 16 -> EIO board
                        */
                        unsigned int BoardRev:5;
```

```
                        /* True  -> Old Z80 protocol */
                        /* False -> New Z80 protocol */
                        unsigned int OldZ80:1;

                        /* True for CMU network environment. */
                        /* False for standalone 10MBit net. */
                        unsigned int CMUNet:1;

                        unsigned int Reserved2:2;

                    } Record382;
                } Case380;
        };


/* Boot Information Record */

struct BIRecord
        {
            union
                {
                    short IntBlk[256];
                    struct {
                            VirtualAddress OvlTable[12];
                                /* Overlays */

                            VirtualAddress VP;
                                /* Address of VP table */

                            VirtualAddress PV;
                                /*  " of PV table */

                            VirtualAddress PVList;
                                /*  " of PV list */

                            VirtualAddress Sector;
                                /*  " of sector headers */
```

```
VirtualAddress PCB;
     /*  * of PCB Handles */

VirtualAddress AST;
     /*  * of AST */

VirtualAddress AccentQueue;
     /*  * of Queue headers */

VirtualAddress AccentFont;
     /*  * of Font */

VirtualAddress AccentCursor;
     /*  * of Cursor */

VirtualAddress AccentScreen;
     /*  * of Screen segment */

short ScreenSize;

short FreeVP;
     /* Initial FreeVP */

short FreeAST;

short SchedProc;
     /* High level scheduling process */

short InitProc;
     /* Initial process */

short BootChar;
     /* Character used in boot */

short NumProc;
     /* Number of processes set up */

short StackSize;
```

```
        /* Size of sup. stack in
           pages*/

short GlobalSize;
        /* Size of sup. global area
           in pages*/

short NumSVReg;
        /* Number of regs for SVCall */
        /*** NumSVRegs is obsolete -
             GGR 11/16/81 ***/

short TrapCode;

VirtualAddress TrapArgs;

short MemBoard;
        /* number of K of memory */

VirtualAddress AccentStdCursor;
VirtualAddress AccentRoTemp;
unsigned char DefaultPartitionName[20];
char IgnoreRunFile;
struct MachineInfoRec MachineInfo;
short Filler[50 - (sizeof(ASTRecord)>>1)];
ASTRecord FirstAst;
VirtualAddress EtherIOArea;
VirtualAddress UserPtr;
        /* Start of user process */
struct Type414
          {
            short SV_CS;
            short SV_GP;
            short SV_LP;
            short SV_LocalSize;
            short SV_TrapCount;
            short SV_FirstRN;
            short SV_PC_Vector[122];
          } SVContext;
```

```
                } Record403;
            } Case395;
    };

typedef struct  BIRecord *ptrBIRecord;
```

6.9. BuiltinDefs

File BuiltinDefs provides the necessary typedefs for Matchmaker-generated interfaces.

Code:

```
#define _BUILTINDEFS

#define TRUE 1
#define FALSE 0
#define NULL 0

/* Borrowed from c_type.h */
typedef short int *caddr_t;
typedef int boolean;
typedef short int Boolean;
typedef char Char;
typedef char Character;
typedef long Long;

typedef char Byte;
typedef long Port;
typedef short Short;
typedef Port Port_All;
typedef Port Port_Receive;
typedef Port Port_Ownership;


/*********************************************************
 * Function movebytes:
 *
 *     Abstract: See header file for movebytes.h.
 *
 *********************************************************/

extern void movebytes ();
```

6.10. CfileDefs

File CfileDefs.h provides the facilities that are used to load a process with a C program. These functions are used by the shell and are not of general use to other programs.

See also the document "Pascal Library" in the *Accent Languages Manual.*

Code:

```
#include <accenttype.h>
#include <timedefs.h>

/*
 * Should be the same as the constant in cfiledefs.pas!!!!!
 *
 */
#define CFileVersion -4

typedef String255 LString;
typedef struct first_block
 {
 short file_version;
 int file_size;
 Internal_Time file_time_stamp;
 int symbol_area_size;
 int text_area_size;
 int data_area_size;
 int bss_area_size;
 int dest_addr;
 int start_addr;
 int main_addr;
 int initial_local_size;
 int stack_base_addr;
 int stack_size_in_pages;

 } FirstBlock, *pFirstBlock;
```

```
  /**/
  /* Symbol entry codes
  /**/
#define  PrimaryDef              10
          /**/
          /* Symbol Kinds (kind of (primary) entry codes)
          /**/
#define          PascalProcedure        1  /* UnUsed */
#define          LocalProcedure         2
#define          GlobalProcedure        3
#define          InitializedSymbol      5
#define          LocalLabel             6
#define          UndefinedSymbol        7
                     /* synonyms */
#define          UndefinedGlobal        UndefinedSymbol
#define          DefinedGlobal          InitializedSymbol
#define          DefinedLocal           LocalLabel
#define  OffsetDef               11
#define  ChainHead               12
#define  AbsoluteDef             13
#define  AbsoluteLocalDef        14
#define  LibraryDef              90

  /**/
  /* Area Designators
  /**/
#define  TextState               0
#define  DataState               1
#define  Data2State              2  /* Internal to asm */
           /* synonyms */
#define  PCInText                TextState
#define  PCInData                DataState
#define  PCInData2               Data2State

  /**/
  /* Misc. stuff
  /**/
#define  OFFSETBASE              16000
                    /* Maximum computed offset value. */
```

```
extern int GetAsmlong();
  /*  char ** byteptr;  */

extern char * GetAsmString();
  /*  char **byteptr;
      char *str;
  */

extern unsigned char GetCodeByte();
  /*  char **byteptr  */

extern int SegBaseAddr();
  /*  int segnum  */

extern short SegmentOf();
  /*  int addr  */

extern void RoundTo();
  /*  int *val;
      short bound;
  */

extern short GetCodeWord();
  /*  char **byteptr;  */

extern void GetLibraryDef();
  /*  char **byteptr;
      char *lib_file_name;
      Internal_Time  *lib_ts;
      int *lib_loc;
  */
```

6.11. Cio/CioDefs

Files Cio.h and CioDefs.h provide the user with the standard IO routines for C.

6.11.1. Cio

Include Files:

```
#include <CioDefs.h>
```

6.11.1.1. Function open

Code:

```
extern int open();
  /*  char *name;
      int mode;
  */
```

Abstract:

Opens a file for reading or writing.

Parameters:

name — A pointer to the string of chars which is the filename

mode — An integer to represent the mode. This is defined in Ciodefs.h as ReadOpen=0, WriteOpen=1, and ReadWriteOpen=2.

Returns:

-1 if not successful.

A file descriptor if successful. (A file descriptor is a small positive integer which is used to identify a particular file instead of using a filename.)

6.11.1.2. Function _open

Code:

```
extern int _open();
  /*  char *name;
      int mode;
      int ind;
  */
```

Abstract:

Attempt to open a file on the given index.

Parameters:

name	A pointer to the string of chars which is the filename
mode	An integer to represent the mode. This is defined in Ciodefs.h as ReadOpen=0, WriteOpen=1, and ReadWriteOpen=2.
ind	The index or file descriptor. (A file descriptor is a small positive integer which is used to identify a particular file instead of using a filename.)

Returns:

-1 if not successful

Index or file descriptor if successful

6.11.1.3. Function read

Code:

```
extern int read();
  /*  int find;
      register char *buffer;
      register int nbytes;
  */
```

Abstract:

Read bytes from a file.

Parameters:

find — The file descriptor or index that identifies your file

buffer — The buffer where the data is to come from

nbytes — The number of bytes to be transferred

Returns:

-1 on error and 0 on eof.

6.11.1.4. Function write

Code:

```
extern int write();
  /*  int find;
      register char *buffer;
      int nbytes;
  */
```

Abstract:

Write bytes to a file.

Parameters:

find The file descriptor or index that identifies your file

buffer Bytes of data will be written to this buffer

nbytes The number of bytes to be written out to the file

Returns:

-1 on error and nbytes if successful

6.11.1.5. Function close

Code:

```
extern int close();
  /*  int find;  */
```

Abstract:

Close a file.

Parameters:

find The index or file descriptor for the file to be closed

Returns:

-1 if unsuccessful and 0 if successful.

6.11.1.6. Function Unlink

Code:

```
extern int unlink();
  /*  char *name;  */
```

Abstract:

Delete a file.

Parameters:

name A pointer to the string of chars that is the filename

Returns:

-1 on error and 0 normally.

6.11.1.7. Function lseek

Code:

```
extern int lseek();
  /*  int find;
      long offset;
      int whence;
  */
```

Abstract:

Lseek allows random access of file data. This moves the current position in the file to 'offset positions from whence'.

Parameters:

find The file descriptor

offset The new position desired

Whence The origin from which the offset will be taken

0 = the beginning of the file

1 = the current position

2 = the end of the file

Returns:

-1 if unsuccessful

0 if the file descriptor is type _NullDevice or _TypeScript

The new position if successful

6.11.1.8. Function tell

Code:

```
extern long tell();
  /* int find; */
```

Abstract:

Returns the current position in the file relative to the begining.

Parameters:

find The file descriptor

Result:

Returns -1 on error or the current position if successful.

6.11.1.9. Function creat

Code:

```
extern int creat();
  /*  char *name;
      int protection;
  */
```

Abstract:

Create a file. Protection is ignored.

Parameters:

name The name of the file to be created

protection currently an int not used in the code

Returns:

-1 if not successful

The file descriptor if successful

6.11.1.10. Function setbuf

extern int setbuf(); /* FILE *fp; char *buf; */

Abstract:

This is not for user programs.

6.11.1.11. Function _cleanup

Code:

```
extern void _cleanup();
  /*  no parameters  */
```

Abstract:

Close all files open for writing.

Parameters:

None

Result:

There is no return value.

6.11.1.12. Function ungetc

Code:

```
extern int ungetc();
  /*  char ch;
      FILE *fp;
  */
```

Abstract:

Provides one character of push back for a text stream.

Parameters:

ch The character to be pushed back onto the file

fp The file pointer

Returns:

-1 if unsuccessful or 0 if successful.

6.11.1.13. Function printf

Code:

```
extern int printf();
  /*  char *format;
      int arguments;
      int a1,a2,a3,a4,a5,a6,a7,a8,a9;
      int b1,b2,b3,b4,b5,b6,b7,b8,b9;
  */
```

Abstract:

Formatted output - where the output is stdout.

Parameters:

format The format control string

arguments The number of arguments depends on the number of conversion specifications in the format string

Returns:

Indeterminate

6.11.1.14. Function fprintf

Code:

```
extern int fprintf();
  /*  FILE *fp;
      char *format;
      int arguments;
      int a1,a2,a3,a4,a5,a6,a7,a8,a9;
      int b1,b2,b3,b4,b5,b6,b7,b8,b9;
  */
```

Abstract:

Output formatting with the output sent to the stream specified as the first argument.

Parameters:

fp The file pointer

format The control string

arguments As required by the control string

Returns:

Indeterminate

6.11.1.15. Function sprintf

Code:

```
extern int sprintf();
  /*  char *resultstr;
      char *format;
      int arguments;
      int a1,a2,a3,a4,a5,a6,a7,a8,a9;
      int b1,b2,b3,b4,b5,b6,b7,b8,b9;
  */
```

Abstract:

Formatted output to a string.

Parameters:

resultstr The string to contain the output

format The control string

arguments The argments as needed from the control string

Results:

Indeterminate

6.11.1.16. Function scanf

Code:

```
extern int scanf();
  /*  char *format;
      int arguments;
      int a1,a2,a3,a4,a5,a6,a7,a8,a9;
      int b1,b2,b3,b4,b5,b6,b7,b8,b9;
  */
```

Abstract:

Reading of formatted input text.

Parameters:

format The control string

arguments As specified by the control string

Returns:

The number of successfully matched and assigned input items. If EOF is reached, EOF (-1) is returned.

6.11.1.17. Function fscanf

Code:

```
extern int fscanf();
  /*  FILE *fp;
      char *format;
      int arguments;
      int a1,a2,a3,a4,a5,a6,a7,a8,a9;
      int b1,b2,b3,b4,b5,b6,b7,b8,b9;
  */
```

Abstract:

Reads formatted input text from fp.

Parameters:

fp The file pointer

format The control string

arguments The arguments as needed in the control string

Returns:

The number of successfully matched and assigned input items. If EOF is

reached, EOF (-1) is returned.

6.11.1.18. Function sscanf

Code:

```
extern int sscanf();

/*  char *str;
    char *format;
    int arguments;
    int a1,a2,a3,a4,a5,a6,a7,a8,a9;
    int b1,b2,b3,b4,b5,b6,b7,b8,b9;
*/
```

Abstract:

Reads formatted input text from str.

Parameters:

str The string from which to read text

format The control string

arguments As specified by the control string

Returns:

The number of successfully matched and assigned input items. If EOF is reached, EOF (-1) is returned.

6.11.1.19. Function _puts

Code:

```
extern int _puts();
  /*  register char *str;
      register FILE *fptr;
      int addnl;
  */
```

Abstract:

Routine to write a string to a file. This routine handles calls to puts and fputs (see the macros in stdio.h)

Parameters:

str — The string to be written to the file

fptr — The file pointer

addnl — 1 or 0. If 1, an extra newline character is written to file.

Returns:

Indeterminate

6.11.1.20. Function gets

Code:

```
extern char *gets();
  /*  register char *str;  */
```

Abstract:

Get a string from stdin. Do not put the \n in the output string.

Parameters:

str — The array where the input string is to be stored

Returns:

str if successful

NULL if unsuccessful

6.11.1.21. Function fgets

Code:

```
extern char *fgets();
  /*  register char *str;
      register int max;
      register FILE *fptr;
  */
```

Abstract:

Read a string from the given file. The returned string will be at most max chars long and include the \n if found.

Parameters:

str	The array to contain the new string
max	The maximum number of chars to be read
fptr	The file pointer

Returns:

The pointer to the new string if successful

NULL is returned if eof occurs

6.11.1.22. Function fgetc

Code:

```
extern char fgetc();
  /*  register FILE *fptr;  */
```

Abstract:

Read a character from any file.

Parameters:

fptr The file pointer

Returns:

The character read

6.11.1.23. Function Fillbuf

Code:

```
extern char fillbuf();
  /*  register FILE *fptr;  */
```

Abstract:

This function is not for user programs.

6.11.1.24. Function fputc

Code:

```
extern int fputc();
  /*  char ch;
      register FILE *fptr;
  */
```

Abstract:

Write a character to a file.

Parameters:

ch The character to be written to the file

fptr The file pointer

Returns:

Indeterminate

6.11.1.25. Function putbuf

Code:

```
extern int putbuf();
  /*  char ch;
      register FILE *fptr;
  */
```

Abstract:

This function is not for user programs.

6.11.1.26. Function fopen

Code:

```
extern FILE *fopen();
  /*  char *name;
      char *mode;
  */
```

Abstract:

Open a file for read, write, or append.

Parameters:

name The filename to be opened

mode "r" for read, "w" for write, "a" for append

Returns:

NULL if unsuccessful. Otherwise the file pointer for the opened file.

6.11.1.27. Function freopen

Code:

```
extern FILE *freopen();
  /*  char *name;
      char *mode;
      FILE *fp;
  */
```

Abstract:

Recycle the stream (i.e., fclose followed by an fopen).

Parameters:

name The filename to be associated with the stream

mode "r" for read, "w" for write, "a" for append

fp The stream or file pointer

Returns:

NULL if unsuccessful or the file pointer if successful.

6.11.1.28. Function fread

Code:

```
extern int fread();
  /*  char *ptr;
      int size;
      int nitems;
      FILE *fp;
  */
```

Abstract:

Read a block of data.

Parameters:

ptr	Points to the buffer where the bytes of data are to be stored
size	The size of the items to be read
nitems	The number of items to be read
fp	The file pointer to the opened stream

Returns:

0 if unsuccessful or the number of items read if successful.

6.11.1.29. Function fwrite

Code:

```
extern int fwrite();
  /*  char *ptr;
      int size;
      int nitems;
      FILE *fp;
  */
```

Abstract:

Write a block of data.

Parameters:

ptr Points to the first item in the buffer to be written

size The size of the items

nitems The number of items to be written

fp The file pointer or stream

Returns:

0 if unsuccessful or the number of items written if successful.

6.11.1.30. Function fclose

Code:

```
extern int fclose();
  /*  FILE *fp;  */
```

Abstract:

Close a file.

Parameters:

fp the file pointer

Returns:

-1 if unsuccessful and 0 if successful.

6.11.1.31. Function fflush

Code:

```
extern int fflush();
  /*  FILE *fp  */
```

Abstract:

Empty the buffer to the destination device.

Parameters:

fp	The stream or file pointer

Returns:

-1 if unsuccessful or 0 if successful.

6.11.2. CioDefs

Code:

```
#include <c_types.h>

#define _NumFiles 20

typedef struct _iobuf
 {
 int _type;  /* type of file desc */
 int _fileno;  /* index */
 int _ts;  /* typescript */
 char *_ptr;  /* current position in the file */
 char *_base;  /* base of the buffer */
 int _count;  /* number of chars left in the buffer */
 char *_oldmark;  /* end of file at last seek */
 int _total;  /* total bytes in file */
 short _flags;  /* mode of the file */
 String255 _fname; /* file name */
 } ;
```

```
extern struct _iobuf  _iob[_NumFiles];
extern char fillbuf();

/*
 *  kinds of opens
 */
#define ReadOpen 0
#define WriteOpen 1
#define ReadWriteOpen 2

/*
 * flag settings
 */
#define _ReadMode 1  /* first bit on */
#define _WriteMode 2  /* second bit on */
#define _ReadWriteMode 3 /* both read & write bits on */
#define _Eof 4

#define _UnusedDescriptor 0
#define _TypeScript 1 /* values that the _type field may have */
#define _DiskFile 2
#define _NullDevice 3

#define BUFSIZ 1024
#define EofChar 032    /* End of file character */

#ifndef NULL
#define NULL 0
#endif NULL

#define EOF (-1)
#define FILE struct _iobuf
#define stdin (&_iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])
#define fileno(p) ((p)->_fileno)
#define feof(fp)  ((fp)->_count <= 0)
```

```
#define putchar(ch)      putc(ch,stdout)
#define getchar()        getc(stdin)
#define getc(fp)         ((fp)->_count-- >
                0 ? *(fp)->_ptr++ : fillbuf(fp))
#define putc(ch,fp)      ((fp)->_count-- >
                0 ? (*(fp)->_ptr++ = (unsigned char)
                (ch)) :putbuf(ch,fp))


#define puts(str)            _puts(str,stdout,TRUE)
#define fputs(str,fp)        _puts(str,fp,FALSE)

#define fseek(fp,os,pn) lseek(fileno(fp),os,pn)
#define ftell(fp) tell(fileno(fp))
```

6.12. Cload/Cloaddefs

Files Cload.h and Cloaddefs.h contain routines and definitions for the C loader.

See also the document "Pascal Library" in the *Accent Languages Manual.*

6.12.1. Cload

Code:

```
#include <accenttype.h>
#include <sesamedefs.h>
#include <cloaddefs.h>


extern int CLoadProcess();
  /*  char *file_name;
      int **file_inmem;
      int *file_size;
      Port proc;
      int load_debug;
  */
```

6.12.2. CloadDefs

Code:

```
#include <timedefs.h>

typedef struct descr_rec
 {
 short version;
 int file_addr;
 int main_addr;
 } DescrRecord, *pDescrRecord;
```

```
#define  DESCRIPTORLOC 01000
#define  DESCRSIZE 512  /* Bytes... probably should even
                           be even. */
#define  DESCRWORDSIZE (DESCRSIZE DIV 2)

/*
 * From acb.dfs
 *
 */
#define  ACBSAVESTACK   10


#define  rRetAddr1 01   /* 1st level micro return code */
#define  rRetAddr2 02   /* 2nd level micro return code */
#define  rVPC  05   /* Program Counter */
#define  rTP  06   /* Top (of stack) Pointer */
#define  rAP  07   /* Activation Pointer */
#define  rGP  010  /* Global Pointer. Now part of
                      procedure return val */
#define  rLP  011  /* Local Pointer */
#define  rRN  012  /* Routine Number. Now part of
                      procedure return val */
#define  rCS  013  /* Code Segment */
#define  rSS  014  /* Stack Segment */
#define  rBkp  015  /* Break Point */
#define  rExcCS  016  /* Exception handler code segment */
#define  rExcGP  017  /* Exception handler global pointer */
#define  rOvlDesc 020  /* Encoded overlay number and entry */

  /* MicroKernel  state is above registers in MicroContext
     block: */
#define  rTrapCode 96
#define  rTrapLsw 97
#define  rTrapMsw 98
#define  rEStkCount 99
#define  rVMState 100
#define  rRegCount 101
#define  rVMCnt  102
#define  rVMDstByte 103
```

```
#define  rVMDstMsw 104
#define  rVMDstLsw 105
#define  rVMSrcByte 106
#define  rVMSrcMsw 107
#define  rVMSrcLsw 108
#define  rVMMsw  109
#define  rVMStatus 110  /* BPC in bits 8..11 */
#define  rResAddr 111


/* Originally in cload.h */
#define CLoadNotCFile -1
```

6.13. Clock

File Clock.h provides a quick 60-hz clock routine.

Code:

```
extern long IOGetTime();
  /*  no parameters  */
```

6.14. Command Parsing Files

The files CommandDefs, CommandParse, CommandParseDefs, and ExtraCmdParse provide routines intended to ease the task of developing utilities that conform to the "standard" command syntax conventions. Command parsing is explained in detail in the document "Pascal Library" in the *Accent Languages Manual.*

6.14.1. CommandDefs

File CommandDefs.h contains definitions for the command structure passed between Accent programs.

Code:

```
/**/
/* Error General Return values
/**/

#define CmdParse_Error_Base     4200

#define ErBadSwitch             CmdParse_Error_Base  +  1
#define ErBadCmd                CmdParse_Error_Base  +  2
#define ErNoSwParam             CmdParse_Error_Base  +  3
#define ErNoCmdParam            CmdParse_Error_Base  +  4
#define ErSwParam               CmdParse_Error_Base  +  5
#define ErCmdParam              CmdParse_Error_Base  +  6
#define ErSwNotUnique           CmdParse_Error_Base  +  7
#define ErCmdNotUnique          CmdParse_Error_Base  +  8
#define ErNoOutFile             CmdParse_Error_Base  +  9
#define ErOneInput              CmdParse_Error_Base  +  10
#define ErOneOutput             CmdParse_Error_Base  +  11
#define ErIllCharAfter          CmdParse_Error_Base  +  12
#define ErBadQuote              CmdParse_Error_Base  +  13
#define ErAnyError              CmdParse_Error_Base  +  14

#define ParseInternalFault      CmdParse_Error_Base  +  15
```

```
#define ParseWordTooLong          CmdParse_Error_Base  +  16
#define PrIlChInSwName            CmdParse_Error_Base  +  17
#define PrIlChInSwVal             CmdParse_Error_Base  +  18
#define PrIlChInEnvNam            CmdParse_Error_Base  +  19
#define PrIlChInQuoted            CmdParse_Error_Base  +  20
#define ParseOnlyCmdAllowed       CmdParse_Error_Base  +  21
#define PrIlChInInRed             CmdParse_Error_Base  +  22
#define PrIlChInOutRed            CmdParse_Error_Base  +  23
#define PrIlChInShPara            CmdParse_Error_Base  +  24

typedef char *Character_Pool;
typedef char *pCharacter_Pool;
typedef long Char_Pool_Index;


typedef struct cmdblock{
        int WordCount;              /* number of words */
        int WordDirIndex;           /* byte index into word
                                       dictionary */
        char * WordArrayPtr;        /* pointer to block that holds
                                       chars, dir */
        int WordArray_Cnt;
        } CommandBlock;

extern CommandBlock Null_CommandBlock();
```

6.14.2. CommandParse

File CommandParse.h provides a number of routines to help with command parsing.

Include Files

```
#include <CommandParseDefs.h>
```

6.14.2.1. Function InitParseDriverTable

Code:

```
extern void InitParseDriverTable ();
  /*  no parameters  */
```

Abstract:

An internal routine to set-up the parsing DFA table.

T, an internal routine, is used to merge new transitions into the DFA table.

Parameters:

None

Returns:

None.

Side Effects:

The parsing DFA table is initialized.

6.14.2.2. Function Null_CommandBlock

Code:

```
extern CommandBlock Null_CommandBlock ();
  /*  no parameters  */
```

6.14.2.3. Function StrLong

Code:

```
extern void StrLong ();
  /*  no parameters  */
```

6.14.2.4. Function InitCommandParse

Code:

```
extern void InitCommandParse ();
  /*  no parameters  */
```

Abstract:

Initializes the parser by marking the parsing DFA tables as un-initialized. The Parser, when first invoked, will notice that the parsing tables need to be initialized and will do so, thus delaying a possibly lengthy initialization process until the data is actually needed.

Parameters:

None

Returns:

None

Side Effects:

Resets the private global table inited flag to FALSE.

6.14.2.5. Function InitCmdFile

Code:

```
extern void InitCmdFile ();
  /*  pCommand_File_List *inF;  */
```

Abstract:

Initializes inF to be a valid Text File corresponding to the keyboard. This must be called before any other command_file routines. The application should then read from inF^.cmdFile. E.g.,

fgets(string, length, inFile^.cmdFile); or

while(! feof(inFile^.cmdFile)) ...

Use popup only if inF^.next = NULL (means no cmd File). Is a fileSystem file if not inF->isCharDevice. InF will never be NULL. The user should not modify the pCommand_File_List pointers; use the procedures provided.

Parameters:

InF Set to the new command list

6.14.2.6. Function OpenCmdFile

Code:

```
extern GeneralReturn OpenCmdFile ();
  /*  pWord_String FileName;
      pCommand_File_List *inF;
  */
```

Abstract:

This function prepares a command file for use by the standard C I/O routines. The user should give OpenCmdFile the word as parsed by one of the parsing routines contained in this CommandParse module. The application is expected to ensure that the command file has appeared in a correct context for that application. These checks might include ensuring that no other words appeared within the input line containing the command file. This function maintains a stack of command files so that command files may contain other command files. Be sure to call InitCmdFile before calling this procedure.

Assumptions: InitCmdFile has been called.

Parameters:

FileName Name of the command file to be opened. The application is responsible for ensuring that the file appears in a correct context.

inF List of command files. This was originally created by InitCmdFile

and is maintained by these routines. If FileName is a valid file, a new entry is put on the front of inF describing it. If there is an error, then inF is not changed. In any case, inF will always be valid.

Returns:

Returns a GeneralReturn code describing the success/failure of the Open.

6.14.2.7. Function ExitCmdFile

Code:

```
extern void ExitCmdFile ();
  /* pCommand_File_List *inF; */
```

Abstract:

Removes top command file from list. Call this whenever the end of a command file is reached.

Suggested use: While feof(inF^.cmdFile) ExitCmdFile(inF);

Assumptions: inF must not be NULL.

Calls: close is used to detach the file from the stream I/O package. fopen is used to re-open the backstop, if necessary. Free is used to free the node's memory.

Parameters:

inF — List of command files. It must never be NULL. The top entry is removed, except when attempting to remove last command file when it is simply re-initialized to be the backstop default input file. It is OK to call this routine even when at the last entry of the list.

Results:

One element is popped off of the command stack, inF.

6.14.2.8. Function ExitAllCmdFiles

Code:

```
extern void ExitAllCmdFiles ();
  /*  pCommand_File_List *inF;  */
```

Abstract:

Removes all command file from the given list. Use when a fatal error has been found or upon receipt of either a SigLevel2Abort or a SigLevel3Abort in order to reset all command files.

Assumptions: inF must not be NULL.

Calls: ExitCmdFile is used to perform an exit on each file of the list.

Parameters:

inF — List of command files. It must never be NULL. All entries but the last are removed.

Results:

inF is reset to a single list node attached to the backstop default input file.

6.14.2.9. Function DstryCmdFiles

Code:

```
extern void DstryCmdFiles ();
  /* pCommand_File_List *inF;  */
```

Abstract:

Removes all command files from the given list. All entries are removed and inF is set to NULL. InitCmdFile must be called again before any command file

stack routines may be used.

Calls: ExitCmdFile is used to release the files. Close and free are used to release the backstop file.

Parameters:

inF List of command files to be released

Results:

inF is set to NULL.

6.14.2.10. Function DestroyCommandParse

Code:

```
extern void DestroyCommandParse ();
  /*  no parameters  */
```

Abstract:

Deallocates the parsing DFA table.

Parameters:

None

Returns:

None

Side Effects:

Resets the private global table inited flag to FALSE.

6.14.2.11. Function AllocCommandNode

Code:

```
extern pCommand_Word_List AllocCommandNode ();
  /*  Word_Type WordClass;
      char * WordString;
  */
```

Abstract:

Allocates a new node to be inserted into the parsed data structures. User is responsible for establishing the correct linkages. This routine merely creates a node with the correct string text value for the word.

Calls: Malloc, to allocate memory.

Parameters:

WordClass Class desired for the new node

WordStringText of the new word

Returns:

Returns a pointer to the newly allocated node.

6.14.2.12. Function DestoyCommandList

Code:

```
extern void DestroyCommandList ();
  /*  pCommand_Word_List *argList;  */
```

Abstract:

Deallocates a parsed data structure.

Calls itself, recursively. Free is used to perform the actual deallocation.

Parameters:

argList Pointer to the structure to be released

Results:

argList is set to NULL.

6.14.2.13. Function AlwaysEof

Code:

```
extern void AlwaysEof ();
  /*  pCharacter_Pool *ChPool;
      Char_Pool_Index *PoolLength;
  */
```

Abstract:

Support routine for callers of ExerciseParseEngine. This routine sets up a character pool which contains an end-of-file marker. This will signal the parsing activity to stop.

Parameters:

ChPool Pointer to the pool to contain the end-of-file marker

PoolLength Length of the eof buffer (always set to 1 by this routine)

Results:

Returns an unbounded character pool containing an end-of-file marker.

6.14.2.14. Function ExerciseParseEngine

Code:

```
extern GeneralReturn ExerciseParseEngine ();
  /*  pCharacter_Pool ChPool;
      Char_Pool_Index PoolLength;
      void (*ReadPool)();  / (Pool,PLen) /
                           / pCharacter_Pool *Pool; /
                           / Char_Pool_Index *PLen; /
      pCommand_Word_List *inputs;
      pCommand_Word_List *outputs;
      pCommand_Word_List *switches;
  */
```

Abstract:

This routine is the Parser. It is called by all other routines which perform parsing. It scans the given character pool (augmented if necessary by reading another pool) and constructs the parsed data lists of inputs, outputs, and switches.

Assumptions: InitCommandParse has been called.

Calls: ReadPool, a procedure parameter to this routine, is used to read the next pool of characters upon exhaustion of the current pool. EnvMgr is called to obtain objects from the Environment. Malloc/Free are used to manage memory. Routines from Spice_String are used to manipulate strings.

Parameters:

ChPool — First hunk of characters to be parsed. This parameter may be NIL; in which case NIL parse structures will be Returned.

PoolLength — Length of the first pool of characters. This parameter may be zero; in which case NIL parse structures will be Returned.

ReadPool Procedure to read successive pools of characters, if required by the parsing actions

inputs List to contain the words recognized as inputs

outputs List to contain the words recognized as outputs

switches List to contain the words recognized as switches

Results:

Places the parsed lists into the parameters and returns a GeneralReturn code of either Success if all went well or an error code indicative of the error.

Errors:

The following general return error codes may be discovered (the caller is expected to arrange for the display of these error conditions):

```
ParseInternalFault
ParseWordTooLong
c - PrIlChInSwName     Pascal - ParseIllegalCharInSwName
c - PrIlChInSwVal      Pascal - ParseIllegalCharInSwVal
c - PrIlChInEnvNam     Pascal - ParseIllegalCharInEnvNam
c - PrIlChInQuoted     Pascal - ParseIllegalCharInQuoted
ParseOnlyCmdAllowed
```

6.14.2.15. Function ParseChPool

Code:

```
extern GeneralReturn ParseChPool ();
  /*  pCharacter_Pool ChPool;
      Char_Pool_Index PoolLength;
      pCommand_Word_List *inputs;
      pCommand_Word_List *outputs;
      pCommand_Word_List *switches;
  */
```

Abstract:

Parses the given unbounded pool of characters.

Calls: ExerciseParseEngine.

Parameters:

ChPool Pointer to the beginning of the pool to be parsed

PoolLength
Number of characters in the pool

inputs List to contain the input words

outputs List to contain the output words

switches List to contain the switches

Results:

Sets the three lists and returns an GeneralReturn code indicative of the parse results.

Errors:

Those of ExerciseParseEngine

6.14.2.16. Function ParseCommand

Code:

```
extern GeneralReturn ParseCommand ();
  /*  pCommand_Word_List *inputs;
      pCommand_Word_List *outputs;
      pCommand_Word_List *switches;
   */
```

Abstract:

Transforms the word list passed into the program into the parsed data lists of input, outputs, and switches.

The program name is normally passed to the program as the first word in the list. This word is stripped and ignored by ParseCommand.

Assumptions: First word in the list passed in by the Shell is the program name and is to be ignored.

Parameters:

inputs — List to contain the input words

outputs — List to contain the output words

switches — List to contain the switches

Results:

Sets the three lists and returns a GeneralReturn code indicative of the parse results.

6.14.2.17. Function WordifyPool

Code:

```
extern GeneralReturn WordifyPool ();
  /*  pCharacter_Pool ChPool;
      Char_Pool_Index PoolLength;
      CommandBlock *WordStruct;
  */
```

Abstract:

Transforms the given unbounded pool of characters into a word list suitable for passing to a Spawn'd child process.

Calls: ExerciseParseEngine. ValidateMemory is used to allocate storage for

the word list.

Parameters:

ChPool Pointer to the beginning of the pool to be 'wordified'

PoolLength
Number of characters in the pool

WordStruct
Buffer to contain the word list description

Results:

Sets the WordStruct and returns an GeneralReturn code indicative of the wordification results.

Errors:

Those of ExerciseParseEngine

6.14.2.18. Function InitWordSearchTable

Code:

```
extern void InitWordSearchTable ();
  /*  pWord_Search_Table *table;
      Boolean CaseSensitive;
  */
```

Abstract:

Creates an empty search table for use in word identification.

Calls: New is used to allocate the new search table.

Parameters:

table Will be set to the address of the new search table

CaseSensitive
: TRUE if the words in this table should retain their capitalization during the identification processing.

Returns:

The address of a new search table is returned in 'table'.

6.14.2.19. Function AddSearchWord

Code:

```
extern void AddSearchWord ();
  /*  pWord_Search_Table table;
      short WordKey;
      char * WordString;
  */
```

Abstract:

Merges the given word into the search table under the given key.

Calls: ConvUpper is used if the table data is not case sensitive. Malloc is used to allocate more table data.

Design: A search item is allocated and prepended to the banner list corresponding to this word.

Exceptions Raised: Will raise Impossible if given illogical parameters.

Parameters:

table
: POINTER to a search table

WordKey
: Identification of the word

WordString
: Text of the word

Returns:

None

Side Effects:

Augments data in the search table.

6.14.2.20. Function DeleteSearchWord

Code:

```
extern void DeleteSearchWord ();
  /*  pWord_Search_Table table;
      char * WordString;
  */
```

Abstract:

Expunges the given word from the search table.

Assumptions: Table must be the result of an InitWordSearchTable call.

Exceptions Raised: Will raise Impossible if given illogical parameters.

Calls: ConvUpper is used if the table data is not case sensitive. free is used to free up memory.

Design: A search item is allocated and prepended to the banner list corresponding to this word.

Parameters:

table POINTER to a search table

WordStringText of the word to be deleted

Returns:

None

6.14.2.21. Function DestroySearchTable

Code:

```
extern void DestroySearchTable ();
  /*  pWord_Search_Table *table;  */
```

Abstract:

Expunges the data for the given search table.

Calls: Uses an internal routine 'DestroySearchItem' to deallocate the nodes on the linked item lists.

Parameters:

table POINTER to the search table to be destroyed

Results:

table is set to NULL.

6.14.2.22. Function UniqueWordIndex

Code:

```
extern short UniqueWordIndex ();
  /*  pWord_Search_Table table;
      pWord_String ptrWordString;
      char * WordText;
  */
```

Abstract:

Locates the given word in the given search table. Returns both the key and the text of the word found.

Assumptions: table has been initialized via InitWordSearchTable and

AddSearchWord.

Parameters:

table _ Search table in which to attempt the identification

ptrWordString
Pointer to the text of the word-prefix to be found

WordText Text of the found word

Returns:

Returns the word-key and the entire text of the identified word if and only if the given word-prefix is in the table and is unique. Will return a WS_NotFound if the word-prefix is not found in the table or a WS_NotUnique if the word-prefix is not unique. In either of the latter error conditions WordText will contain the erroneous word-prefix.

6.14.2.23. Function ConvertPoolToString

Code:

```
extern pCmnd_String ConvertPoolToString ();
  /*  pCharacter_Pool ChPool;
      Char_Pool_Index FirstChar;
      Char_Pool_Index StringLength;
  */
```

Abstract:

Transforms an arbitrary portion of the given unbounded pool of characters into a C String.

Assumptions: The result of a function is to be placed in its local #0.

Parameters:

ChPool Pointer to the unbounded hunk of characters

FirstChar Beginning position (zero based) within ChPool of the desired string

StringLength
Number of characters to be moved from the pool to the string

Returns:

Returns a C String.

6.14.2.24. Function ConvertStringToPool

Code:

```
extern void ConvertStringToPool ();
  /*  char * CnvStr;
      pCharacter_Pool *ChPool;
      Char_Pool_Index *PoolLength;
  */
```

Abstract:

Transforms a C String into an unbounded pool of characters suitable for use by the ParseChPool routine. Is symetric with ConvertPoolToString.

Calls: ValidateMemory is used to grab memory for the hunk of characters.

Design: Uses MVBW QCode to byte copy from the string to the pool.

Parameters:

CnvStr C String to be converted

ChPool Set to the address of the created pool

PoolLength
Number of characters in the resultant pool

Results:

Sets the parameters 'ChPool' and 'PoolLength' to describe the new unbounded hunk of characters.

6.14.2.25. Function DestroyChPool

Code:

```
extern void DestroyChPool ();
  /*  pCharacter_Pool *ChPool;
      Char_Pool_Index *PoolLength;
  */
```

Abstract:

Deallocates storage of a given unbounded pool of characters.

Calls: InvalidateMemory is used to do the deallocation.

Parameters:

ChPool Pointer to the pool to be released; is set to NULL

PoolLength Number of characters to be released; is zeroed

Results:

Both ChPool and PoolLength are modified to describe an empty pool.

6.14.2.26. Function GetIthWordPtr

Code:

```
extern pWord_String GetIthWordPtr();
   /* long i;
      char ** _comargv;
   */
```

Abstract:

Retrieves a pointer to the desired word from the given word list.

Parameters:

i Number (one-based) of the desired word

CmndBlock
Block from which to fetch the word

Returns:

Returns a pointer to the requested word or NULL if the desired number is out of bounds.

6.14.3. CommandParseDefs

File CommandParseDefs.h provides definitions of the separator characters.

```
#include <AccentType.h>
#include <CommandDefs.h>
#include <crt0.h>
#include <stdio.h>
#include <EnvMgrDefs.h>
#include <EnvMgr.h>
/*#include "Except.h"*/
/*#include "Stream.h"*/
/*#include "PascalInit.h"*/
#include <PathName.h>
#include <Spice_String.h>


                        /** DELIMITERS: **/
#define eofChar                     0
     /* "end of file" - may NEVER appear in a word */

#define eolnChar                   012
    /* end of line */

#define single_char_quote          '\\'
    /* quote just the next char */
```

```
#define quoted_text_bracket_char '\''
    /* quote an entire string */

#define env_var_bracket_char     '~'
    /* substitute from the environment */

#define env_quoted_bracket_char  '*'
    /* environ subst into a quoted string */

#define switch_leadin_char       '-'
#define value_marker_char        '='
#define command_file_leadin_char '@'
#define comment_leadin_char      '#'
#define in_out_separator_char    '~'

#define shell_special_args_start '['
    /* these are recognized by */

#define shell_special_args_stop  ']'
    /* the shell only */

#define shell_input_redirect     '<'
#define shell_output_redirect    '>'
    /* parsing routines herein attach */

#define shell_sequential_command ';'
    /* no special meaning to them and */

#define shell_piped_command      '|'
    /* treat them as if they were */

#define shell_parallel_command   '&'
    /* alpha-numerics */


        /* CHARACTERS TO BE APPENDED TO PROMPTS */

#define CmdChar          0200 + 24
        /* use outside command files */
```

```
#define CmdFileChar      0200 + 26
        /* use inside command files */


#define CommandParseVersion "5.8 of 30 Sep 84"
#define MaxCmndString 255


typedef char Cmnd_String[MaxCmndString];
typedef char *pCmnd_String;

/* typedef unsigned char  Word_String[1];
     */ /* DO NOT try to STORE into a Word_String */
/*typedef Word_String     *pWord_String;    */
typedef char * pWord_String;


       /* USED WITH COMMAND FILE NESTING ROUTINES */


typedef struct Command_File_List *pCommand_File_List;

typedef struct Command_File_List  { /* the command file stack */
        FILE * cmdFile;
        Boolean isCharDevice;
        pCommand_File_List next;
        };


                /* USED WITH PARSING ROUTINES */

#define in_arg           0
#define out_arg          1
#define switch_arg       2
#define switch_value     3
#define command_file     4

typedef byte Word_Type;
```

```
     /* USE AllocCommandNode (SEE BELOW) TO MAKE ONE OF THESE */

typedef struct Command_Word_List
    { /* the structure of a parsed command */

        pWord_String ptrWordString;

        unsigned int DeallocWordString:1;
           /* DO NOT MESS WITH THIS FIELD */

        Word_Type WordClass;

        union {
                struct Command_Word_List * NextArg;
                struct {
                        struct Command_Word_List * NextSwitch;
                        struct Command_Word_List * ValueOfSwitch;
                        struct Command_Word_List * CorrespondingArg;
                        } switcharg;
                } caseWordClass;
        };

typedef struct Command_Word_List *pCommand_Word_List;


           /* USED WITH WORD IDENTIFICATION ROUTINES */

           /* ERRORS RETURNED BY UNIQUEWORDINDEX */

#define WS_NotFound  -1          /* word-prefix not found */
#define WS_NotUnique -2          /* word-prefix not unique */

typedef short * pWord_Search_Table;
        /*structure of table is strictly private*/
```

```
  /* MISCELLANEOUS CHAR POOL MANIPULATION ROUTINES */


#define debugging false

   /* Actions which the parsing DFA may take prior to
   /* transitioning from one state to another. Note that
   /* the order of these has significance. The specific
   /* action to be taken on a particular transition may
   /* be the union of any of these; however, the actions
   /* are executed in a top to bottom fashion. That is,
   /* if both StartArg and AdvanceChar are actions to
   /* occur at a transition then StartArg is executed
   /* first and then AdvanceChar is executed. */


#define StartArg              0
#define StartCmdArg           1
#define StartSwitch           2
#define StartSwValue          3
#define StartEnvVar           4
#define AccumEnvVar           5
#define StoreCurrChar         6
#define FinishArg             7
#define FinishSwitch          8
#define FinishSwValue         9
#define FinishEnvVar         10
#define FinishQuotedEnv      11
#define MarkInOut             12
#define AdvanceChar           13
#define ErrorBadSwitchName   14
#define ErrorBadSwitchValue 15
#define ErrorBadEnvName      16
#define ErrorIllegalQuote    17
#define Finished              18

typedef byte Transition_Actions;
```

```
#define First_Action StartArg
#define Last_Action  Finished

/* Masks used to form the unions of Actions for each
   transition */
#define StartArg_Mask                (int)(1 << StartArg)
#define StartCmdArg_Mask             (int)(1 << StartCmdArg)
#define StartSwitch_Mask             (int)(1 << StartSwitch)
#define StartSwValue_Mask            (int)(1 << StartSwValue)
#define StartEnvVar_Mask             (int)(1 << StartEnvVar)
#define AccumEnvVar_Mask             (int)(1 << AccumEnvVar)
#define StoreCurrChar_Mask           (int)(1 << StoreCurrChar)
#define FinishArg_Mask               (int)(1 << FinishArg)
#define FinishSwitch_Mask            (int)(1 << FinishSwitch)
#define FinishSwValue_Mask           (int)(1 << FinishSwValue)
#define FinishEnvVar_Mask            (int)(1 << FinishEnvVar)
#define FinishQuotedEnv_Mask         (int)(1 << FinishQuotedEnv)
#define MarkInOut_Mask               (int)(1 << MarkInOut)
#define AdvanceChar_Mask             (int)(1 << AdvanceChar)
#define ErrorBadSwitchName_Mask      (int)(1 << ErrorBadSwitchName)
#define ErrorBadSwitchValue_Mask     (int)(1 << ErrorBadSwitchValue)
#define ErrorBadEnvName_Mask         (int)(1 << ErrorBadEnvName)
#define ErrorIllegalQuote_Mask       (int)(1 << ErrorIllegalQuote)
#define Finished_Mask                (int)(1 << Finished)

   /* the States of the> DFA */

#define initial_state 0
#define accumulate_argument_chars 1
#define first_arg_quoted_char 2
#define midst_arg_quoted_char 3
#define arg_quoted_string 4
#define arg_env_var 5
#define arg_quoted_env 6
#define accumulate_switch_chars 7
#define first_switch_quoted_char 8
#define midst_switch_quoted_char 9
#define switch_quoted_string 10
#define switch_env_var 11
```

```
#define switch_quoted_env 12
#define accumulate_value_chars 13
#define first_value_quoted_char 14
#define midst_value_quoted_char 15
#define value_quoted_string 16
#define value_env_var 17
#define value_quoted_env 18
#define env_for_quoted_arg 19
#define env_for_quoted_switch 20
#define env_for_quoted_value 21
#define skip_leading_switch_blanks 22
#define skip_trailing_switch_blanks 23
#define skip_leading_value_blanks 24
#define final_state 25

typedef byte State_Index;


typedef struct Transition_Descriptor { /* a DFA transition */
        char TransitChar;
        State_Index NextState;
        struct Transition_Descriptor * NextTransit;
        long ActionFlags;
        };

typedef struct Transition_Descriptor *pTransition_Descriptor;

typedef struct State_Entry  { /* banner nodes for each state */
        pTransition_Descriptor FirstTransit, LastTransit;
        };

typedef struct Search_Item {
        Cmnd_String str;
        short key;
        struct Search_Item * collide_link;
        };

typedef struct Search_Item *pSearch_Item;
```

```
typedef struct Internal_Search_Table
            { /* what a Word Search Table really is */
        Boolean CaseSense;
        pSearch_Item ch_headers[128-32+1];
        };

typedef struct Internal_Search_Table *pInternal_Search_Table;

Boolean TableInited;
struct State_Entry State_Table[final_state+1];
              /* the DFA itself */


/* RAISED BY CATASTROPHIC INTERNAL FAULTS */
/** exception impossible **/
```

6.14.4. ExtraCmdParse

This module provides procedures to help with command lines and yes-or-no questions.

Include Files and Defines:

```
#include <CommandParseDefs.h>
#include <CommandParse.h>


/* anomolous conditions reported by GetCmd and GetShellCmd */

/* first input word not found in search table */
#define Cmd_NotFound WS_NotFound

/* first input word-prefix not unique */
#define Cmd_NotUnique WS_NotUnique

/* user line was empty */

/* user line contained no inputs but maybe has switches */
#define Cmd_EmptyCmdLine -3
```

```
#define Cmd_NotInsMaybeSwitches -4

/* some error has occurred -- code is in ErrorGR */
#define Cmd_SomeError -5

/* indicators returned by GetConfirm */
#define Confirm_YES 1
#define Confirm_NO 2
#define Confirm_Switches 3

/* amount to indent prompts for params */
#define Prompt_Indention_String "    "


#define Version "V2.5c of 15 May 85"
```

6.14.4.1. Function GetCmd

Code:

```
extern short GetCmd ();
   /* Cmnd_String prompt;
      pWord_Search_Table SearchTable;
      Cmnd_String CmdName;
      pCommand_File_List *inF;
      pCommand_Word_List *inputs;
      pCommand_Word_List *outputs;
      pCommand_Word_List *switches;
      GeneralReturn *ErrorGR;
   */
```

Abstract:

GetCmd obtains a command input from the top file on the given file list. It then processes that command in several ways. First, it parses the command. Next it examines the parsed structures to determine if a command file reference was the first (and only) item in the command. If a command file is found, that file is prepended to the command file list and a null command is signalled to the caller. If no command file is found, then an attempt to locate

the first input argument in the given search table is made. The results of that search are returned to the caller along with the text of the first argument and the lists of inputs (sans first arg), outputs, and switches.

Assumptions: 1) InitCmdFile has been called; 2) InitCommandParse has been called; 3) SearchTable has been appropriately initialized via InitWordSearchTable and AddSearchWord calls; 4) The current default output file has been rewritten.

Calls:

```
ParseCommand        - to actually parse the user's word list.
OpenCmdFile         - nests commmand files.
UniqueWordIndex     - lookup ptrCmdName in SearchTable.
DestroyCommandList - releases temporary memory consumed by
                        a command file input spec.
```

Parameters:

Prompt Prompt string to print for the user. Do not put the prompt separator (>) on the end of the prompt; GetCmd will do that for you. If reading from a command file, GetCmd changes the prompt appropriately. Prompt is always displayed on the current default output file.

SearchTable
Word search in which to attempt to locate the user's first input

CmdName Buffer to contain the text of the user's first input word

inF Command file list from which to read the user's command and on which to nest command files (if any)

inputs List to contain the user's input words

outputs List to contain the user's output words

switches List to contain the user's switch selections

ErrorGR GeneralReturn code indicating status of the parse processing

Returns:

Returns the UniqueWordIndex of WordKey for CmdName in SearchTable or

```
Cmd_NotFound              - first input word not found
                            in SearchTable
Cmd_NotUnique             - first input word not unique
Cmd_EmptyCmdLine          - command line was empty
Cmd_NotInsMaybeSwitches - no inputs appeared on the line
                            but there may be switches
Cmd_SomeError             - error was discovered (ErrorGR
                            contains error code)
```

Side Effects:

Will nest a new command file onto the InF list if given a command file specification as the user's first word (via OpenCmdFile).

Errors:

1) All those found by GetParsedUserInput.

2) Illegal command file spec -- more than just the command file on the input line.

6.14.4.2. Function GetShellCmd

Code:

```
extern short GetShellCmd ();
   /* pWord_Search_Table SearchTable;
      Cmnd_String CmdName;
      pCommand_File_List *inF;
      pCommand_Word_List *inputs;
      pCommand_Word_List *outputs;
      pCommand_Word_List *switches;
```

```
    GeneralReturn *ErrorGR;
*/
```

Abstract:

This routine is similar to GetCmd except that it works on the command line specified to the Shell. It is used by programs that use GetCmd so that the Shell command line may be parsed in a similar manner. Command files are handled by GetShellCmd in a manner like GetCmd.

Assumptions: 1) InitCmdFile has been called; 2) InitCommandParse has been called; 3) SearchTable has been appropriately initialized via InitWordSearchTable and AddSearchWord calls.

Calls:

```
ParseCommand        - to actually parse the user's word list.
OpenCmdFile         - nests commmand files.
UniqueWordIndex     - lookup ptrCmdName in SearchTable.
DestroyCommandList - releases temporary memory consumed by
                        a command file input spec.
```

Parameters:

SearchTable
: Word search in which to attempt to locate the user's first input

CmdName Buffer to contain the text of the user's first input

inF Command file list on which to nest command files (if any)

inputs List to contain the user's input words

outputs List to contain the user's output words

switches List to contain the user's switch selections

ErrorGR GeneralReturn code indicating status of the parse processing

Returns:

Identical to GetCmd. Viz: returns the UniqueWordIndex of WordKey for CmdName in SearchTable or

```
Cmd_NotFound              - first input word not found in
                            SearchTable
Cmd_NotUnique             - first input word not unique
Cmd_EmptyCmdLine          - command line was empty
Cmd_NotInsMaybeSwitches - no inputs appeared on the line
                            but there may be switches
Cmd_SomeError             - error was discovered (ErrorGR
                            contains code)
```

Side Effects:

Will nest a new command file onto the InF list if given a command file specification as the user's first word (via OpenCmdFile).

Errors:

1) All those found by ParseCommand (in module CommandParse).

2) Illegal command file spec -- more than just the command file on the input line.

6.14.4.3. Function GetParsedUserInput

Code:

```
extern GeneralReturn GetParsedUserInput ();
   /* pCmnd_String prompt;
      pCommand_File_List *inF;
      pCommand_Word_List *inputs;
      pCommand_Word_List *outputs;
      pCommand_Word_List *switches;
   */
```

Abstract:

GetParsedUserInput obtains a command input from the top file on the given command list. It then parses that command and returns to the caller the lists of inputs, outputs, and switches. This routine is intended for those applications in which the rules for parsing user input into words is desired but for which the application desires to perform all processing of the parsed words. (That is, use this if you don't want the command-file and word-identification effects of the GetCmd routine).

Assumptions: 1) InitCmdFile has been called; 2) InitCommandParse has been called; 3) The current default output file has been rewritten.

Calls:

1) Standard C I/O routines are used to interact with the user.
2) ValidateMemory/InvalidateMemory are used to allocate/deallocate a chunk of memory to be used for the unbounded char pool used by parsing.
3) ExerciseParseEngine - parses the user command line.
4) ExitCmdFile - closes inF if found.

Parameters:

Prompt	Prompt string to print for the user. Prompt string is displayed as is, no modifications made. Prompt is always displayed on the current default output file.
inF	Command file list from which to read the user's command
inputs	List to contain the user's input words
outputs	List to contain the user's output words
switches	List to contain the user's switch selections

Returns:

Returns a GeneralReturn code indicating either Success, if all went well, or an appropriate error code.

Side Effects:

If eof is discovered on the input file it will be closed (via ExitCmdFile).

Errors:

Those found by ExerciseParseEngine

6.14.4.4. Function GetConfirm

Code:

```
extern short GetConfirm ();
   /* Cmnd_String prompt;
      short def;
      pCommand_Word_List *switches;
   */
```

Abstract:

Handles a question that is to be answered Yes or No where the answer should come from the keyboard. Prompt followed by default (if any) is printed. Prompt may be null. If illegal input is typed, GetConfirm re-asks but doesn't use prompt.

Assumptions: 1) InitCmdFile has been called; 2) InitCommandParse has been called; 3) The current default output file has been rewritten.

Calls:

1) Standard C I/O routines are used to interact with the user.
2) [Show/Remove]WindowAttentionFlag (in WindowUtils) are used to manipulate the Icon flag.
3) ExerciseParseEngine - parses the user confirmation line.
4) The Word Searching routines of CommandParse are used to

identify a Yes/No confirmation entry.
5) DestroyCommandList is used to clean up memory we may have allocated.

Parameters:

prompt Prompt to display for question. Prompt is always displayed on the current default output file.

default Index of the default answer:

Confirm_YES = true or yes;

Confirm_NO = false or no; other numbers mean no default.

switches - set to NIL or a list of switches specified. Be sure to handle the switches first since one might be HELP.

Returns:

Confirm_YES if true or yes.

Confirm_NO if false or no.

Confirm_Switches if naked return when no default and switches <> NIL. This means that there was no argument but a switch was entered If an answer is still needed, the application should re-call GetConfirm.

6.14.4.5. Function GetCharacterPool

Code:

```
extern void GetCharacterPool ();
 /*  Cmnd_String prompt;
     FILE * InputFile;
     pCharacter_Pool *ChPool;
     Char_Pool_Index *PoolLength;
 */
```

Abstract:

GetCharacterPool should be used by those applications which wish to perform their own command input line parsing. This routine will interact with the user to obtain an unbounded raw pool of characters.

Assumptions: The current default output file has been rewritten.

Calls:

1) standard c io routines to display prompt
2) ValidateMemory is used to allocate a chunk of memory to be used for the unbounded char pool used by parsing.

Note: The caller of GetCharacterPool should also call DestroyChPool in order to free up memory allocated here.

Parameters:

Prompt — Prompt string to print for the user GetCharacterPool displays this string exactly as given; no changes made. Prompt is always displayed on the current default output file.

InputFile — File from which to read the pool of chars

ChPool — Pointer to the pool read

PoolLength — number of characters read

Returns:

All results returned via the parameters.

6.15. Configuration/ConfigurationDefs

Files Configuration.h and ConfigurationDefs.h contain the routines and definitions for returning PERQ configuration information. Configuration.h accesses the Boot Information Block for information; it will not work if the process has no access to physical memory.

6.15.1. Configuration

Code:

```
#include <configurationdefs.h>

extern Cf_IOBoardType Cf_IOBoard ();
  /*  no parameters  */


extern Cf_MonitorType Cf_Monitor ();
  /*  no parameters  */


extern Cf_Z80Type Cf_OldZ80 ();
  /*  no parameters  */


extern Cf_NetworkType Cf_Network ();
  /*  no parameters  */
```

6.15.2. ConfigurationDefs

Code:

```
#include <stdio.h>
#include <BootInfo.h>

#define Cf_CIO 0                /* Perq1 */
#define Cf_EIO 1
typedef char Cf_IOBoardType; /* Perq2 */
```

```
#define Cf_Landscape 0
#define Cf_Portrait 1
typedef char Cf_MonitorType;


#define Cf_CMUNet 0
#define Cf_10MBitNet 1
typedef char Cf_NetworkType;


#define Cf_NewZ80 0
#define Cf_OZ80 1
typedef char Cf_Z80Type;
```

6.16. ControlStore/ConstrolStoreDefs

6.16.1. ControlStore

File ControlStore.h exports types defining the format of PERQ micro-instructions and procedures to load and call routines in the control-store. You must have physical memory privileges to use these procedures.

Code:

```
extern void  LoadControlStore();
  /*          int fileindex;
      ( this is the number returned from "open" (not "fopen"))
  */

extern void  LoadMicroInstruction();
  /*    short Adrs;
        MicroInstruction MI;
  */

extern void  JumpControlStore();
  /*  short Adrs;  */
```

6.16.2. ControlStoreDefs

Code:

```
typedef union {                    /* The format of a
                                    /* micro-instruction
                                    /* as produced by the
                                    /* micro-assembler */
        struct {
                short Word1;
                short Word2;
                short Word3;
                } Case0;

        struct {
```

```
                unsigned int Jmp:4;
                unsigned int Cnd:4;
                unsigned int Z:8;
                unsigned int SF:4;
                unsigned int F:2;
                unsigned int ALU:4;
                unsigned int H:1;
                unsigned int W:1;
                unsigned int B:1;
                unsigned int A:3;
                unsigned int Y:8;
                unsigned int X:8;
                } Case1;

        struct {
                unsigned int JmpCnd:8;
                unsigned int Fill1:8;
                unsigned int SFF:6;
                unsigned int ALU0:1;
                unsigned int ALU1:1;
                unsigned int ALU23:2;
                } Case2;

        } MicroInstruction;

typedef struct {                        /* The format of a
                                          /* micro-instruction
         short Adrs;                     /* and its address as
                                          /* produced by the
         MicroInstruction MI;           /* micro-assembler */
        }MicroBinary;

typedef union {                         /* The format of a micro-
                                          /* instruction as needed
                                          /* by the WCS QCode */

        struct {
                short Word1;
                short Word2;
                short Word3;
```

```
        } Case0;

struct {
        unsigned int ALU23:2;
        unsigned int ALU0:1;
        unsigned int W:1;
        unsigned int ALU1:1;
        unsigned int A:3;
        unsigned int Z:8;
        unsigned int SFF:6;
        unsigned int H:1;
        unsigned int B:1;
        unsigned int JmpCnd:8;
        } Case1;

}TransMicro;
```

6.17. Crt0/Crt0Defs

Files Crt0.h and Crt0Defs.h contain definitions of ports that have already been initialized for the user and of items used in process initialization.

6.17.1. Crt0

Code:

```
#include <Crt0Defs.h>

/*
 * These are the things available to users.
 */
extern int ComputingEnvironment;
      /* 0 for Accent 1 for QNIX started processes */
extern int InPorts_Cnt;
      /* Number of Ports passed to process in InPorts array */
extern Port             TimePort;
extern Port             SesPort;
extern Port             EMPort;
extern Port             PMPort;
extern Port             NameServerPort;
extern Port             SapphPort;
extern Port             TypescriptPort;
extern Port             UserWindow;
extern Port             UserTypescript;
extern Boolean          UserWindowShared;
extern PortArray        *InPorts;
```

6.17.1.1. Function crt0

Code:

```
extern int crt0();
  /*  no parameters  */
```

Abstract:

This routine is usually (unless otherwise specified in the lnk command) the first piece of code to be executed for the user process. This code calls init_process, initializes stdin, etc., and then calls the main routine.

NOTE: Not meant to be called from a user program.

6.17.1.2. Function init_process

Code:

```
extern int init_process();
  /*  no parameters  */
```

Abstract:

This routine calls a work routine (which receives the initial message checks and unpacks it) and then initializes connections to the major system servers.

NOTE: Not meant to be called from a user program.

```
/**-------------------------------------------------------
 **  SPECIAL NOTE FOR USERS OF: black_putc, black_puts,
 **  black_printf
 **
 **      The arguments will be displayed in reverse video
 **      starting either at the top left hand corner of
 **      the screen or at the end of the last argument
 **      which was printed with one of these functions.
 **      This text is not under Window management and
 **      therefore will not be refreshed if a window is
 **      displayed on top of the text.  This also means
 **      that display of the text cannot be forward or
 **      backward scrolled for redisplay.
 **
 **      When the text lines are being displayed about
 **      halfway down the screen, the process will pause
 **      and wait for the user to hit a carriage return
 **      before displaying any more text.  The text will
 **      then continue to scroll until another half-screenful
 **      has been displayed and the process will again
 **      pause for a carriage return.  (This display mode
 **      prevents output from scrolling off the screen
 **      before the user has a chance to see it).
 **
 **-------------------------------------------------------*/
```

6.17.1.3. Function black_putc

Code:

```
extern int black_putc();
  /*  char character  */
```

Abstract:

Writes a single Character in reverse-video directly to the screen without using any Typescript or Windowing functions. No buffering is performed, the output is immediately displayed on the screen.

Parameters:

character The single character to be displayed

6.17.1.4. Function black_puts

Code:

```
extern int black_puts();
  /*  char *string;  */n
```

Abstract:

Uses black_putc to display a string on the screen. (See black_putc).

Parameters:

string The string to be displayed

6.17.1.5. Function black_printf

Code:

```
extern int black_printf();
  /*  char *format;
      int arguments;
      int a1,a2,a3,a4,a5,a6,a7,a8,a9;
      int b1,b2,b3,b4,b5,b6,b7,b8,b9;
  */
```

Abstract:

Performs the equivalent of the printf function, but in reverse-video without buffering or Typescript or Windowing functions.

Parameters:

Similar to printf. Format characters are:

c Print the following argument as an ASCII character

s	Print following argument which is a pointer to a string
d	Argument is a word, print as a signed decimal number
o	Word unsigned octal
x	Word unsigned hexadecimal
b	Word unsigned binary

6.17.1.6. Function _bufprintf

Code:

```
extern int _bufprintf();
  /*    char *buffer;
        register char *format;
        register int *parguments;
  */
```

Abstract:

Using the format and parguments parameters, create a string to be displayed and place it in the buffer pointed to by the first param.

Parameters:

buffer — Points to a buffer large enough to hold print arguments

format — String describing how the arguments are to be formated. Valid format control characters are described in the black_printf documentation above.

parguments — Arguments to be displayed

Returns:

The number of characters placed in the buffer

6.17.1.7. Function exit

Code:

```
extern int exit();
  /*  int status;  */
```

Abstract:

Calls _cleanup (in cio) to close any open files associated with this process, and then calls _exit(status);

Parameters:

status Exit status to be returned

6.17.1.8. Function _exit

Code:

```
extern int _exit();
  /*  int status;  */
```

Abstract:

Terminates process without closing open files.

Returns:

Does not return. Process is terminated.

6.17.1.9. Function EnablePrivs

Code:

```
extern GeneralReturn   EnablePrivs();
  /*  Port Proc;  */
```

Abstract:

Enable physical memory access and supervisor access for a process.

Parameters:

Proc The Kernel port of the process to affect

Result:

Success Privileges have been enabled

Failure Privileges not affected. This shouldn't ever occur unless you violate the precondition.

Precondition:

This procedure will only work if Proc is 'KernelPort' (i.e., you are modifying your own privileges), or if the process to be affected is suspended.

6.17.1.10. Function DisablePrivs

Code:

```
extern GeneralReturn   DisablePrivs();
  /*  Port Proc;  */
```

Abstract:

Disable physical memory access and supervisor access for a process.

Parameters:

Proc The Kernel port of the process to affect

Result:

Success Privileges have been disabled

Failure Privileges not affected. This shouldn't ever occur unless you violate the precondition.

Precondition:

This procedure will only work if Proc is 'KernelPort' (i.e., you are modifying your own privileges), or if the process to be affected is suspended.

6.17.1.11. Function GetShellArgs

Code:

```
extern void GetShellArgs();
  /*     int * MyArgcAddr;
         long * MyArgvAddr;
   */
```

Abstract:

Returns argc and argv through parameters. The caller of this routine should declare two variables as:

```
int MyArgC;
char ** MyArgV;
```

and then should call GetShellArgs as:

```
GetShellArgs(&MyArgC, &MyArgV);
```

Upon return MyArgC and MyArgV will contain the argument count and a pointer to the argument list respectively. Calling main as

```
main(MyArgC,MyArgV)
    int MyArgC;
    char ** MyArgV;
```

would result in identical values being assigned to MyArgC and MyArgV. Use of this routine was intended for use by library functions (such as CommandParse) which need to have access to the command line, but which cannot depend upon main being called with argc, argv as parameters.

Parameters:

MyArgcAddr
: Pointer to the arg count

MyArgvAddr
: Pointer to the arg list pointer

Returns:

Returns void. Parameters have been changed to contain argc and argv for the process.

6.17.2. Crt0Defs

Code:

```
#include <AccentType.h>

#define InitMsgID       32896     /* new one for sapphire */

/* Magic Port Indices */
#define TimeIndex        0
#define SesameIndex      1
#define SapphIndex       2
#define TypescriptIndex  3
#define PMIndex          4
#define MsgIndex         5
#define FirstUserIndex   6

/* WARNING:  There is code that depends on the above numbers
 *           being zero based and on FirstUserIndex being
 *           both the first free index AND the number of
 *           required indicies.
 */

#define QNIX 1
#define ACCENT 0

struct InitialMsg {
                    Msg                 Head;
```

```
                    TypeType            DefaultType;
                    String255           DefaultInputName;
                    String255           DefaultOutputName;
                    TypeType            PassPortType;
                    ptrPortArray        PassedPorts;
                    TypeType            UWinSharedType;
                    Boolean             UWinShared;
                    TypeType            EMType;
                    Port                EMPort;
                    TypeType            WindowType;
                    Port                UserWindow;
                    TypeType            TsType;
                    Port                UserTs;
                    TypeType            WordCountType;
                    int                 WordCount;
                    TypeType            WordDirIndexType;
                    int                 WordDirIndex;
                    TypeType            WordArrayPtrType;
                                             /* Long form */
                    short               WordArrayPtrEltSize;
                    short               WordArrayPtrTName;
                    int                 WordArray_Cnt;
                    int                 *WordArrayPtr;
                    TypeType            ComputeEnvType;
                    int                 ComputingEnvironment;
};

#define InitMsgSize     (sizeof(struct InitialMsg))

#define DESCRRECORDLOC  ((struct DescrRecord *)01000)
/*
 * This structure is placed at DESCRRECORDLOC by the loader.
 * From it we can find the address of 'main' (MainAddr),
 * as well as the pascal string which is the command line.
 */
struct DescrRecord {
                    Integer             Version;
                    long int            FileAddr;
                    int                 (*MainAddr)();
```

```
};
```

6.18. C_types

File C_Types defines Pascal-like types for C interfaces.

Code:

```
#include        <SysType.h>


/*
 *      Basic unit of memory which we can point to
 */

#if              PERQ
typedef          short int *      caddr_t;
#else
typedef          char *           caddr_t;
#endif

/*
 *      Provide boolean type
 */

typedef          int              boolean;
                                   /* For the Die-Hard C folks. */
typedef          short int        Boolean;
                                   /* For Pascal Compatibility */
#ifndef TRUE
#define  TRUE    1
#endif

#ifndef FALSE
#define  FALSE   0
#endif


/*
 *      Model for Perq Pascal "String" type...
 *              Macro DefineString creates a definition
```

```
 *              of a string with specified length; 'caller'
 *              is responsible for naming the resulting type.
 */

#define         DefineString(NameOfType, SizeOfString) \
typedef         union \
               { \
               unsigned char    _StrSize; \
               char             _StringChars[SizeOfString+1]; \
               } \
                                  NameOfType

DefineString (String80, 80);
DefineString (String255, 255);

typedef         String80        String;

/*
 *      Other built-in types
 */

typedef         char            int8;
typedef         short int       int16;
typedef         long  int       int32;

typedef         char            Character;
typedef         char            Char;
typedef         int16           Integer;
typedef         int32           Long;

#ifndef         NULL
#define         NULL            0
#endif          NULL
```

6.19. Doprnt/DoprntDefs

Files Doprnt.h and DoprntDefs.h contain the routine and definitions for the internals to printf and the output conversion routines.

6.19.1. Doprnt

Include Files:

```
#include <DoprntDefs.h>
```

6.19.1.1. Function _doprnt

Code:

```
extern void _doprnt();
  /*   register
       char    *format;
       int     *arglist;
       register
       FILE    *stream;
  */
```

Abstract:

The internals to printf and the output conversion routines.

Parameters:

format — The format control string

arguments — The number of arguments depends on the number of conversion specifications in the format string

stream — The output stream

Returns:

Return is void.

6.19.2. DoprntDefs

Code:

```
#include <C_Types.h>

#if SPOONIX
#include <stdio.h>
#endif
#if NATIVE_ACCENT
#define FILE char
#define putc(thing,place)    *(place)++=(thing)
#define BUFSIZ 1024
#endif

#define   ROUNDING  TRUE

/*
 * To build a "standalone" version of doprnt, which doesn't
 * require much from the stdio and gen libraries, turn on
 * the STANDALONE conditional.
 */

#if  STANDALONE
#define   _doprnt  _black_doprnt
#undef  putc
#define  putc(c,f) black_putc (c)
#endif STANDALONE

#if  METER
int     _doprnt_count = 0;
#endif METER
```

6.20. EnvMgr/EnvMgrDefs

Files EnvMgr.h and EnvMgrDefs.h contain the routines and definitions used by the environment manager calls.

See the document "The Environment Manager" in the *Accent Programming Manual.*

6.20.1. EnvMgr

Code:

```
#include <AccentType.h>
#include <EnvMgrDefs.h>

extern void InitEnvMgr () ;
  /*  Port RPort;  */

extern GeneralReturn GetEnvVariable ( );
  /*  Port ServPort;
      Env_Var_Name Name;
      Env_Var_Scope SearchScope;
      Env_Variable  * Variable;
      Long  * Variable_Cnt;
      Env_Var_Type  * VarType;
      Env_Var_Scope  * ActualScope;
  */


extern GeneralReturn SetEnvVariable ( );
  /*  Port ServPort;
      Env_Var_Name Name;
      Env_Var_Type VarType;
      Env_Var_Scope VarScope;
      Env_Variable Variable;
      Long Variable_Cnt;
  */

extern GeneralReturn ScanEnvVariables ( );
```

```
  /*  Port ServPort;
      Env_Var_Scope SearchScope;
      Env_Scan_List  * EnvScanList;
      Long  * EnvScanList_Cnt;
  */

extern GeneralReturn ResolveSearchList ( );
  /*  Port ServPort;
      Env_Var_Name Name;
      Boolean FirstOnly;
      Env_Variable  * Variable;
      Long  * Variable_Cnt;
      Boolean  * FirstDefined;
  */


extern GeneralRetur CopyEnvConnection ( );
  /*  Port ServPort;
      Port OldConnection;
      Port  * NewConnection;
  */


extern GeneralReturn EnvDisconnect ( );
  /*  Port ServPort;  */
```

6.20.2. EnvMgrDefs

Code:

```
#include <SesameDefs.h>

#define Env_Element_Size 255

DefineString(Env_Element,Env_Element_Size);

typedef Env_Element  *Env_Element_Array;
typedef Env_Element  *Env_Variable;
```

```
/*
 * A Searchlist name embedded in a searchlist string is followed by
 * a Searchlist_Separator character.
 */

#define  Searchlist_Separator  ':'

/*
 * Env_Var_Name:  The name string for an environment variable.
 *       The syntax of the name is the same as for an arbitrary
 *       entry name in the name server.
 */

#define Env_Variable_Size Entry_Name_Size

DefineString(Env_Var_Name,Env_Variable_Size);

/*
 * Env_Var_Type:  The environment variable type values.
 */

typedef         short           Env_Var_Type;
#define         Env_String      0
#define         Env_SearchList  1

/*
 * Env_Var_Scope:  Flag specifying whether to find environment
 *       variable in the local table, global table, or using
 *       the normal method of local and then global.
 */

typedef         short           Env_Var_Scope;
#define         Env_Normal      0
#define         Env_Local       1
#define         Env_Global      2

/*
 * Env_Scan_List:  A list of environment variable names, types,
 * and scopes.
```

```
 */

typedef struct
        {
          Env_Var_Name          VarName;
          Env_Var_Type          VarType;
          Env_Var_Scope         VarScope;
        } Env_Scan_Record;

typedef Env_Scan_Record  *Env_Scan_Array;
typedef Env_Scan_Record  *Env_Scan_List;

/*
 * Error return values for Environment Manager.
 */

#define Env_Error_Base          1600

#define EnvVariableNotFound     (Env_Error_Base + 1)
#define WrongEnvVarType         (Env_Error_Base + 2)
#define BadSearchlistSyntax     (Env_Error_Base + 3)
#define SearchlistLoop          (Env_Error_Base + 4)
#define FirstItemNotDefined     (Env_Error_Base + 5)
```

6.21. Errfile

The file Errfile.h contains the functions used in error detection (because C has no exception mechanism). Generally, GRError is invoked from a Matchmaker interface.

Include Files:

```
#include <stdio.h>
```

6.21.1. Function GRError

Code:

```
extern short GRError();
  /*  short grerr;  */
```

Abstract:

This function is generally invoked from MatchMaker interfaces for C. This is C's temporary solution for exceptions - it simply prints the GR value to stderr.(The exit will terminate the current process.)

Parameters:

grerr The General Return generated during the last system call.

6.21.2. Function WipeOut

Code:

```
extern short WipeOut();
  /*  char * s;  */
```

Abstract:

Print a message to stderr and then exit.

Parameters:

s The string to be printed

6.22. Ftoa

Include Files:

```
#include <C_Types.h>
```

6.22.1. Function ftoa

Code:

```
extern void ftoa();
  /*    register
        char    *buffer;
        register
        double    dbl;
        int     places;
        boolean    exp_style;
        boolean    rounding;
  */
```

Abstract:

File Ftoa.h provides floating point output conversion, including exponential notation.

Parameters:

buffer	The character array to contain the ascii string after conversion
dbl	The floating point number
places	The number of decimal places desired in the output string
exp_style	1 for exponential format; otherwise 0
rounding	1 if rounding is to occur and 0 if not

6.23. GetBits

File GetBits.h contains routine definitions which allow the caller to extract or insert bits into specific fields in a 32-bit entry.

6.23.1. Function GetBits

Code:

```
extern int GetBits();
 /*  long BitRecs;
     int Loc;
     int Cnt;
 */
```

Abstract:

Returns "Cnt" bits starting at location "loc" in BitRecs.

```
Example:     X= 0753 (binary 111101011)
              GetBits(X, 3, 2) returns  2 (binary 010)
```

Parameters:

BitRecs — The 32 bit word to get the bits from

Loc — The low order bit to start getting the bits from

Cnt — The number of bits to get

Returns:

An Integer value with the Bits requested

6.23.2. Function SetBits

Code:

```
extern long SetBits();
  /*  long BitRecs;
      int Loc;
      int Cnt;
      int Val;
  */
```

Abstract:

Set Cnt Bits of BitRecs (starting at bit Loc) to be Val

Example: X = 02345 (binary 010011100101)
SetBits(X, 4, 3, 07) returns 02375
(010011111101)

Parameters:

BitRecs	The 32bit work to put the bits into
Loc	The low order bit to start putting the bits at
Cnt	The number of bits to put into BitRecs
Val	The bits to put into BitRecs. These should be in the low order bits of Val.

Returns:

BitRecs with Cnt bits Val inserted at Loc

Warnings:

If Val has some bits set past bit Cnt, the warning: "SetBits bits out of range" is issued. Execution continues. (e.g. if Val = binary 10111 and Cnt = 4, the

warning will be issued).

6.24. IFileDefs

File IFileDefs.h contains the internal definitions for the file system.

Code:

```
#include <stdio.h>
#include <OldTime.h> /* Using TimeStamp */
#include <AccentType.h> /* Using SegID */
#include <SesameDefs.h>
#include <SesDiskDefs.h>

#define SimpleNameSize 25
#define PartialNameSize 80
#define FIBlk -1 /* Block number of the File Information
                    Block */

/**/
/* The (partial) Filename in the FSDataEntry is compatible
    with POS.
**/
#define POS_Dir_Separator '>'

/**/
/* The internal access rights flags are arranged so that
*  a mask of 0 (from POS or a system running without access
*  protection) is Owner read-and-write,  World read-only.
**/

#define Owner_NO_Read_Access  001
#define Owner_NO_Write_Access 002
#define World_NO_Read_Access  004
#define World_Write_Access    010

typedef short Access_Type;
#define Read_Access  0
#define Write_Access 1
```

```
#define Owner_Access 2


typedef long BlockNumber; /* Legal file block numbers */

typedef char POSSimpleName[SimpleNameSize + 1];
                    /* one component of file name */
typedef char PartialPathName[PartialNameSize + 1];
                    /* file name including all directories */

/*
/* POS style file header.
/* Must be 52 words long.
*/

typedef struct /* FSDataEntry */
      {
       short FileBlocks;
              /* word 0:      Size of file in blocks */

       unsigned int FileBits:13;
              /* word 1<0:11>Number of bits in last blk */

       unsigned int FileSparse:1;
              /* word 1<12>  true if can be sparse */

       unsigned int Unused:2;
       TimeStamp FileCreateDate;      /* word 2-3 */
       TimeStamp FileWriteDate;       /* word 4-5 */
       TimeStamp FileAccessDate;      /* word 6-7 */
       unsigned int FileType:2;
          /* word 8<0:1> File(0), directory(3) */

       unsigned int FileRights:4;
             /* word 8<2:5> Used to be type
              * IntAccessFlags */

       unsigned int FileOwner:10;
             /* word 8<6:15> Used to be type Bit10 */
```

```
        long FileDataFormat;
              /* word 9-10 */

        PartialPathName Filename;
              /* word 11 */

      } FSDataEntry;
            /* end FSDataEntry struct */

typedef struct FSDataEntry *ptrFSDataEntry;



/**/
/* Entry for a port
**/
typedef struct
      {
         Port              PortName;
         User_ID           PortOwner;
         unsigned int      PortRights:4;
      } PortRecord;
typedef struct PortRecord *pPort;

/**/
/* Entry for a foreign (sesamoid) file system.
**/
typedef struct
      {
         Port ForeignPort;
         APath_Name ForeignPrefix;
         User_ID ForeignOwner;
         unsigned int ForeignRights:4;
      } ForeignRecord;
typedef struct ForeignRecord *pForeign;
```

```
/**/
/* internal record for entries
* this MUST be 2 words long to fit in Directory segment!
**/

typedef struct
     {
        union
            {
               SegID SId;
                  /* Segment ID of file or directory */

               short *DirPtr;
                  /* pointer to memory image of
                     directory */

               pPort EPort;
                  /* IPC port */

               pForeign EForeign;
                  /* A foreign sesamoid record */

            } Case546;
     } EntryVal;

typedef struct
     {
        Entry_Type EType;
        EntryVal ID;
     } EntryRec;



/* Definitions for POS-compatible directory structure */

/*
      The following definitions tell how many entries
      there are in the three pieces of the random index.
      The first piece (Direct) are blocks whose
```

```
        DiskAddresses are actually contained in the Random
        Index (which is part of the FileInformationBlock).
        The second section has a list of blocks each of
        which contain 128 Disk Addresses of blocks in the
        file, forming a one level indirect addressing
        scheme.  For very large files, the third section
        (DblInd) has DiskAddresses of blocks which point
        to other blocks which contain 128 DiskAddresses
        of blocks in the file, forming a two level indirect
        scheme.
*/

#define DIRECTSIZE 64
     /* Entries in FIB of blocks directly accessable */

#define INDSIZE    32
     /* Entries in FIB of 1 level indirect blocks */

#define DBLINDSIZE  2
     /* Entries in FIB of 2 level indirect blocks */

#define FILESPERDIRBLK 16
     /* 256 div WordSize(DirEntry) */

#define HASHFRAMESIZE  31
    /* Hash frame for directory */

/*
        A directory is an ordinary file which contains
        SegIDs of files along with their names.
        Directories are hash coded by file name to make
        lookup fast.  They are often sparse files (i.e.,
        contain unallocated blocks between allocated blocks).
        The file name is a POSSimpleName, since a directory
        can only contain entries for files within the
        directory itself.
*/
```

```
typedef struct
     {
        unsigned int InUse:1;
             /* true if this DirEntry is valid */

        unsigned int Deleted:1;
             /* true if entry deleted but not expunged */

        unsigned int Archived:1;
             /* true if entry is on backup tape */

        unsigned int EType:13;
             /* valid range is 0 .. 017777 (octal) */
             /* ZERO on disk */
             /* EntryType in memory (if not file ) */
        EntryVal ID;                 /* Entry value */
        POSSimpleName Filename;
     } DirEntry;


typedef union
    {                    /* The following is the format of the
                            FileInformationBlock; the FIB
                            has Logical Block -1: */

      struct /* FIB */
       {
           FSDataEntry FSData;

         /* The Random Index is a hint of the DiskAddresses
            of the blocks that form the file.  It has three
            parts as noted above.  Notice that all three
            parts are always there, so that even in a very
            large file, the first DIRECTSIZE blocks can be
            located quickly.  The blocks in the Random index
            have logical block numbers that are negative.
            The logical block number of Indirect[0] is -2
            (the FIB is -1).  The last possible block's number
```

```
        is -(INDSIZE+DBLINBDSIZE+1).
    */

    DiskAddr Direct[DIRECTSIZE];
    DiskAddr Indirect[INDSIZE];
    DiskAddr DblInd[DBLINDSIZE];

    SpiceSegKind SegKind;

    short NumBlksInUse;
          /* segments can have gaps, block n may
             exist when block n-1 has never been
             allocated.  NumBlksInUse says how
             many data blocks are actually used
             by the segment.
                                */
    short LastBlk;
          /* Logical Block Number of largest block
             allocated.
                                */
    DiskAddr LastAddr;
          /* DiskAddr of LastBlk */

    short LastNegBlk;
          /* Logical Block Number of largest pointer
             block allocated
                                */
    DiskAddr LastNegAddr
          /* Block number of LastNegBlk*/

  } FIB; /* end FIB struct */

   /* The following is the format of the directory
      header as stored in the Name Server: */

struct /* DirHead */
  {
   FSDataEntry DFSData;
        /* same as in -1 block */
```

```
        struct DirBlock * Addr;
             /* address of the 0 block in memory */

        SegID Segger;
             /* Segment number of directory on disk
              * Segger used to be named "Segment",
              * which conflicted with a define */

        Port PPort;
             /* Partition port for directory */

        short NumPages;
             /* last block allocated */

        short HighBlock;
             /* highest block mapped into memory */

        struct DirBlock *forptr;

        struct DirBlock *backptr;

        unsigned int NoSegment:1;
            /* not in disk if true */

        unsigned int Attached:1
           /* may only be Detached, not Deleted */

       } DirHead;  /* end DirHead struct */

        /* The following is the format of a block
           of a Directory: */

         DirEntry Entry[FILESPERDIRBLK];

    } DirBlock; /* end DirBlock Union */


typedef struct DirBlock *pDirBlock;
```

```
/*
* File types put into the FileType field of a POS file.
* The only one used by Accent is DirFile.
**/

#define UnknownFile  0
#define DirFile      3
#define ExDirFile    4
#define SwapFile     17  /* a file used for swapping
                            by compiler or editor;
                            length not set */
#define BadFile      18  /* created by the scavenger */
```

6.25. IO/IODefs

Files IO.h and IODefs.h contain the definitions and routines for the IO System. See the document "The IO System" in the *Accent Programming Manual.*

6.25.1. IO

Code:

```
#include <IODefs.h>
#include <AccentUser.h>

extern char *IO_Version ( );
  /*  ServerNamePort SERVERPORT;  */

extern GeneralReturn OpenIO ( );
  /*  ServerNamePort SERVERPORT;
      ServerIOPort *IOPORT;
      UserEventPort USERPORT;
  */

extern GeneralReturn CloseIO ( );
  /*  ServerIOPort IOPORT;  */

extern GeneralReturn SyncIO ( );
  /*  ServerIOPort IOPORT;
      IOCommand COMMAND;
      pCmdBlk CMDBLK;
      long CMDBLK_CNT;
      pDataBuf * DATABUF;
      long *DATABUF_CNT;
      Long DATATRANSFERCNT;
      Long TIMEOUT_ARG;
      IOStatusBlk * STATUS;
  */


extern void ASyncIO ( );
```

```
  /*  ServerIOPort IOPORT;
      IOCommand COMMAND;
      pCmdBlk CMDBLK;
      long CMDBLK_CNT;
      pDataBuf DATABUF;
      long DATABUF_CNT;
      Long DATATRANSFERCNT;
      Long TIMEOUT_ARG;
      Boolean NOTIFYONCOMPLETION;
  */

extern Boolean XIOCompletion ( );
  /*  caddr_t InP;
      void (*IOCompletion)();
   */

/*      USER WRITTEN ROUTINE PASSED TO XIOCOMPLETION     */
/* void IOCompletion(TransID, DBuf, DBuf_Cnt, Stat)      */
/*    long TransID;                                        */
/*    pDataBuf DBuf;                                       */
/*    long DBuf_Cnt;                                       */
/*    IOStatusBlk Stat;                                    */
/*                                                           */


extern Boolean XAsyncData ( );
  /*  caddr_t InP;
      void (*AsynchData)();
   */

/*      USER WRITTEN ROUTINE PASSED TO XASYNCDATA         */
/*void AsyncData(SeqNum, DBuf, DBuf_Cnt, Stat)            */
/*    long SeqNum;                                         */
/*    pDataBuf DBuf;                                       */
/*    long DBuf_Cnt;                                       */
/*    IOStatusBlk Stat;                                    */
/*                                                           */
```

```
extern Boolean XAttention ( );
  /*  caddr_t InP;
      void (*Attention)();
   */

/*       USER WRITTEN ROUTINE PASSED TO XATTENTION        */
/*void Attention(AtnCause)                                 */
/*    long AtnCause;                                       */
/*                                                         */


extern Boolean XDistress ( );
  /*  caddr_t InP;
      void (*Distress)();
   */

/*       USER WRITTEN ROUTINE PASSED TO XDISTRESS         */
/*void Distress(TransID, DBuf, DBuf_Cnt, Stat)             */
/*    long TransID;                                        */
/*    pDataBuf DBuf;                                       */
/*    long DBuf_Cnt;                                       */
/*    IOStatusBlk Stat;                                    */
/*                                                         */


extern Boolean XAcknowledge ( );
  /*  caddr_t InP;
      void (*Acknowledge)();
   */

/*       USER WRITTEN ROUTINE PASSED TO XACKNOWLEDGE      */
/*void Acknowledge(TransID)                                */
/*     long TransID;                                       */
/*                                                         */


extern Boolean XServerFull ( );
  /*  caddr_t InP;
      void (*ServerFull)();
```

```
   */

/*      USER WRITTEN ROUTINE PASSED TO XSERVERFULL      */
/*void ServerFull(TransID)                                */
/*     long TransID;                                      */
/*                                                        */


extern Boolean IOAsynch ( );
  /*  caddr_t InP;  */

extern void InitIO ( );
  /*  Port RPort;  */
```

6.25.2. IODefs

Code:

```
#if 0
#include <BuiltInDefs.h>
#else
#include <C_Types.h>
#include <OldBuiltInDefs.h>
#endif

#include <AccentType.h>

typedef char IOString_80[80];

typedef char * pDataBuf;

typedef char * pCmdBlk;

typedef Port ServerNamePort;

typedef Port ServerIOPort;

typedef Port UserEventPort;
```

```
typedef short IOCommand;

typedef struct {
    Msg Head;
    Integer Body[1024];
} IOMessage;


typedef IOMessage *pIOMessage;


typedef struct {
                    Bit8  RegNum;
                    Bit8 RegVal;

                  }   GPIBWriteRegister;



typedef struct  {
                    Bit8 RegNum;
                    Bit8 RegVal;

                  }   SIOWriteRegister;



typedef struct  { /* Options - Bitmap to select cmd actions */
                    unsigned int SetInt0Mask:1;
                    unsigned int SetInt1Mask:1;
                    unsigned int OmitBusConfig:1;
                    unsigned int OmitUnListen:1;

                    union {
                            struct{
                                unsigned int HoldOffOnEOI:1;
                                unsigned int unused2:3;
                                   }   IODevReadCmd;
```

```
                        struct{
                            unsigned int OmitGoToStandby:1;
                            unsigned int WaitOnData:1;
                            unsigned int ForceEOI:1;
                            unsigned int unused1:1;
                               }   IODevWriteCmd;

                      }   IORorW;

                Bit8 Filler;

                Bit8 Int0Mask;
                  /* Mask for 9914 Interrupt Reg 0 */

                Bit8 Int1Mask;
                  /* Mask for 9914 Interrupt Reg 1 */

                Bit8 PrimAddr;
                 /* Primary Address of Device      */

                Bit8 SecAddr;
                 /* Secondary Address of device    */

                Bit8 ReadCount;
                 /* IODevRead                       */

              }   GPIBDevCmdHead;



typedef short DensityType;


typedef struct  {
            int CmdIDTag;
            union { /* Based on IODevice */

                    /* No Device */
```

```
union { /* No Device for generic access */
       char * Cmdbyte;  /* Byte Access */
       short * CmdWord; /* Word Access */
       char * WriteReg; /* Register Acess */
        } NoDevice;


/*  GPIB  */
union { /* Based on IOCommand */

        /* IOSetAttention */
        Boolean GPIBEnableATN;

        /* IOSetStream    */
        struct {
                 Boolean GPIBEnableStream;
                 short GPIBBlockingFactor;
                } GPIBSS;

        /* IOSetBufferSize */
        int GPIBBufferSize;

        /* IOWriteRegisters */
        GPIBWriteRegister * GPIBWriteReg;

        /* IODevRead, IODevWrite */
        GPIBDevCmdHead GPIBDevCmdBlk;

      }  GPIBDev;


/* RS232A, RS232B */
union { /* Based on IOCommand */

        /* IOSetAttention */
        Boolean RS232EnableATN;

        /* IOSetStream */
```

```
            struct {
                    Boolean RS232EnableStream;
                    short   RS232BlockingFactor;
                    } RSSS;

            /* IOSetBufferSize (only for
               RS232A) */
            int RS232BufferSize;

            /* IOSetBaud */
            struct {
                     Bit8 RS232TxBaud;
                     Bit8 RS232RxBaud;
                    } RSSB;

            /* IOWriteRegisters */
            SIOWriteRegister *RS232WriteReg;
          }      RSDev;


/* Speech */
union { /* Based on IOCommand */

            /* IOSetAttention */
            Boolean SpeechEnableATN;

            /* IOSetBufferSize */
            int SpeechBufferSize;

            /* IOSetBaud */
            short SpeechTxRate;

            /* IOWriteRegisters */
            SIOWriteRegister *SpeechWriteReg;
          }  SpDev;


/* Floppy */
```

```
                union { /* Based on IOCommand */

                        /* IOSetAttention */
                        Boolean FloppyEnableATN;

                        /* IOSetDensity */
                        DensityType FloppyDensity;

                        /* IORead, IOWrite, IOFormat,
                           IOSeek, IORecalibrate,
                           IOReadID, IOSenseDrive */
                        struct { Bit8  FloppyUnit;
                                 Bit8  FloppyHead;

                                 /* IORead, IOWrite, */
                                 /* IOFormat, IOSeek */
                                 Bit8 FloppyCylinder;

                                 /* IORead, IOWrite ->
                                    Sector Number */
                                 /* IOFormat ->
                                    FloppyFmtData   */
                                 Bit8 SectorOrFormat;

                               } FUHRec ;
                      } FlpU;

              } IODevU;

        } IOCmdBlk;


typedef IOCmdBlk *pIOCmdBlk;


typedef struct {
    Bit8 IntStat0;
```

```
    Bit8 IntStat1;
    Bit8 IntAddrStat;
    Bit8 IntBusStat;
    Bit8 IntAddrSwch;
    Bit8 IntCmdPass;
    Bit8 CurAddrStat;
    Bit8 CurBusStat;
    Bit8 CurAddrSwch;
    Bit8 CurCmdPass;
} GPIBSenseStatus;


typedef  struct  {
                    /* Read General Status - SIO Chip's
                       Read Register #0 */
                    unsigned int RxCharAvailable:1;
                    unsigned int IntPending:1;
                    unsigned int TxBufferEmpty:1;
                    unsigned int DCD:1;
                    unsigned int SyncHunt:1;
                    unsigned int CTS:1;
                    unsigned int TransmitUnderRun:1;
                    unsigned int BreakAbort:1;

                    /* Read Special Condition - SIO Chip's
                       Read Register #1 */
                    unsigned int AllSent:1;
                    unsigned int Residue:3;
                    unsigned int ParityError:1;
                    unsigned int RxOverRun:1;
                    unsigned int CrcFramingError:1;
                    unsigned int EndOfFrame:1;
                 }     SIOSenseStatus;


typedef short FloppyResultType;
```

```
/*
 *
 *      The Pascal version of iodefs uses bit fields for
 *      StatusReg0 and StatusReg3.  However because of
 *      word alignments in C, StatusReg0 and StatusReg3
 *      are declared as a single Bit8 field.  The user
 *      will therefore have to use macros to get at the
 *      single bits according to the following:
 *
 *      When FloppyResultType = HeadChange or CmdResults,
 *      StatusReg0 is allocated as follows:
 *              Unit : 2 bits
 *              Headr: 1 bit
 *              NotReady : 1 bit
 *              EquipFault : 1 bit
 *              SeekEnd : 1 bit
 *              IntrCode : 2 bits
 *
 *
 *      When FloppyResultType = DriveSense, StatReg3 is
 *      allocated.
 *
 *      StatReg3:   Unit : 2 bits
 *                  Headr: 1 bit
 *                  TwoSided : 1 bit
 *                  AtTrack0 : 1 bit
 *                  DriveReady : 1 bit
 *                  WriteProtected : 1 bit
 *                  DriveFault : 1 bit
 *
 *
 *      The following defines can be used to fill all of the
 *      1 bit fields listed above.
 */

/* StatReg0 */
#define Headr 4
#define NotReady 8
#define EquipFault 16
```

```
#define SeekEnd 32

/*  StatReg3  */
#define TwoSided 8
#define AtTrack0 16
#define DriveReady 32
#define WriteProtected 64
#define DriveFault 128


/*  The following macros can be used to insert and extract
    bits in the fields of the status registers which are
    longer than 1 bit.  */

#define _bUnit 0
#define _sUnit 2

#define _bIntrCode  6
#define _sIntrCode 2

/* Macro to insert into Unit of StatReg0 and StatReg3  */
#define _Unit(x)  ((x&(~(-1<< _sUnit)))   << _bUnit)

/* Macro to extract from Unit of StatReg0 and StatReg3  */
#define _X_Unit(x) ((x>> _bUnit)  & (~(-1 << _sUnit)))

/* Macro to insert into IntrCode of StatReg0  */
#define _IntrCode(x)  ((x & (~(-1<< _sIntrCode))) << _bIntrCode)

/* Macro to extract from IntrCode of StatReg0  */
#define _X_IntrCode(x) ((x>> _bIntrCode) & (~(-1<< _sIntrCode)))



typedef
    struct  {
            Bit8 StatusType;    /* Only 2 bits are needed */

             Bit8 StatReg0or3;  /* Represents Reg3 only if
```

```
                                Status Type is DriveSense,
                                otherwise it's Reg0. */

        union {
                Bit8 PresentCylinder;
                        /* FloppyResultType = HeadChange */

                struct {

                        struct{ /* Status Reg 1 */
                            unsigned int NoAddrMark:1;
                            unsigned int NotWritable:1;
                            unsigned int NoData:1;
                            unsigned int Unused1:1;
                            unsigned int Overrun:1;
                            unsigned int DataError:1;
                            unsigned int Unused2:1;
                            unsigned int TrackEnd:1;
                                 } StatReg1;

                        struct{ /* Status Reg 2 */
                            unsigned int NoDataAddrMark:1;
                            unsigned int BadTrack:1;
                            unsigned int ScanFail:1;
                            unsigned int ScanHit:1;
                            unsigned int WrongCylinder:1;
                            unsigned int DataCRCError:1;
                            unsigned int ControlMark:1;
                            unsigned int Unused3:1;
                                 } StatReg2;

                        Bit8  CylinderID;
                        Bit8  HeadID;
                        Bit8  SectorID;
                        Bit8  SectorSizeCode;
                        }    CmdResultsRec;
                       /* FloppyResultType = CmdResults */

                } HeadorCmdRec;
```

```
        } FloppyResultStatus;


typedef struct  IOSSBLK {
            Bit8 StatusCnt;

            union {  /* IODevice */

                  GPIBSenseStatus GPIBStatus; /*  GPIB  */

                  SIOSenseStatus  SIOStatus;  /*  RS232A,
                                                  RS232B,
                                                  Speech  */

                  FloppyResultStatus FloppyStatus; /* Floppy */

                  char * StatusByte;      /*  Generic Access  */

                  short * StatusWord;     /*  Generic Access  */

                  char SByte[12];         /*  Generic Access  */

                  } IOBlk;
          }     IOSenseStatusBlk;



typedef struct  IOSBlk {
                           long  CmdIDTag;
                           short HardStatus;
                           short SoftStatus;
                           long  CmdBytesTransferred;
                           long  DataBytesTransferred;
                           IOSenseStatusBlk DeviceStatus;
```

```
} IOStatusBlk;


#define IOBaseMsgID 4000
#define IOSuccess 101
#define IOErr 4000
#define IOUndefinedError 4001
#define IOTimeOut 4002
#define IODeviceNotFree 4003
#define IOInvalidIOPort 4004
#define IOBadUserEventPort 4005
#define IOBadPortReference 4006
#define IOIllegalCommand 4007
#define IOBadCmdBlkCount 4008
#define IOBadDataByteCount 4009
#define IOBadRegisterNumber 4010
#define IONotEnoughRoom 4011
#define IOBadBaudRate 4012
#define IONoDataFound 4013
#define IOOverRun 4014
#define IOParityError 4015
#define IOFramingError 4016
#define IOCircBufOverFlow 4017
#define IOEndOfFrame 4018
#define IOEndOfInput 4019
#define IOBadSectorNumber 4020
#define IOBadCylinderNumber 4021
#define IOBadHeadNumber 4022
#define IOUndeterminedEquipFault 4023
#define IODeviceNotReady 4024
#define IOMissingDataAddrMark 4025
#define IOMissingHeaderAddrMark 4026
#define IODeviceNotWritable 4027
#define IOSectorNotFound 4028
#define IODataCRCError 4029
#define IOHeaderCRCError 4030
#define IOBadTrack 4031
#define IOCylinderMisMatch 4032
```

```
#define IODriveReadyChanged 4033
#define IONotEnoughData 4034
#define IOBadBufferSize 4035
#define IOExclusionFailure 4036
#define IOCancelled 4037
#define IOStartIOComplete 4038
#define IOMustBeAsynchronous 4039
#define IOBadBlockingFactor 4040
#define IOMustEnableAsyncIO 4041
#define IOServerFull 4042
#define IOAborted 4043
#define BigByteSize 2048
#define BigWordSize 1024
#define BigLongSize 512
#define IOSense 0
#define IOReset 1
#define IOWriteRegisters 2
#define IOFlushInput 3
#define IOFlushOutPut 4
#define IORead 5
#define IOWrite 6
#define IOWriteEOI 7
#define IOReadHiVol 8
#define IOWriteHiVol 9
#define IODevRead 10
#define IODevWrite 11
#define IOSetBaud 12
#define IOSetStream 13
#define IOSetAttention 14
#define IOAbort 15
#define IOSuspend 16
#define IOResume 17
#define IOSeek 18
#define IORecalibrate 19
#define IOFormat 20
#define IOReadID 21
#define IOSenseDrive 22
#define IOSetDensity 23
#define IOSetBufferSize 24
```

```
#define IONullCmd 25
#define SingleDensity 0
#define DoubleDensity 1
#define NoStatus 0
#define HeadChange 1
#define DriveSense 2
#define CmdResults 3
#define RSExt 0
#define RS110 1
#define RS150 2
#define RS300 3
#define RS600 4
#define RS1200 5
#define RS2400 6
#define RS4800 7
#define RS9600 8
#define RS19200 9
```

6.26. Itoa

File Itoa.h contains an integer output conversion routine, which takes a number in "num" and produces the character string representing this number in base "radix."

Include Files:

```
#include <C_Types.h>
```

6.26.1. Function itoa

Code:

```
extern void itoa();
 /*      register char          *buffer;
         register unsigned long int  num;
         register unsigned int  radix;
         boolean      issigned;
 */
```

Abstract:

See above.

Parameters:

buffer	Points to a char array which is large enough to hold string representation of "num"
num	Holds the value which is to be coverted to its string representation
radix	The base of the number
issigned	TRUE if num is signed, FALSE otherwise

Returns:

Places character string in array pointed to by buffer.

6.27. Keytran/Keytrandefs

Files Keytran.h and Keytrandefs.h contain routines and definitions for the Keytran module passed between Accent programs. Key translation is explained in the document "The Window Manager" in the *Accent Programming Manual.*

6.27.1. Keytran

Include Files:

#include <KeyTranDefs.h> #include <SapphDefs.h> #include <PathName.h>

6.27.1.1. Function AddToKeyTable

Code:

```
extern GeneralReturn AddToKeyTable ();
  /*  RawIn Key1;
      TranOut Key2;
      pKeyTab tab;
      short *ConflictDepth;
  */
```

Abstract:

Adds the new entry to the existing keytranslation TABLE.

Parameters:

Key1 — The UNTRANSLATED event info containing the region and the keystate info.

Key2 — The TRANSLATED event info containing the command, the escape kind and the character.

tab — The pointer to the keytranslation table.

conflictDepth
: If added an entry and it was a conflict, then this depth is returned, otherwise 0.

Returns:

Success or GR value if failed. Returns this conflict depth if there was a conflict.

6.27.1.2. Function ChangeKeyTable

Code:

```
extern GeneralReturn ChangeKeyTable ();
  /*  RawIn OldKey;
      RawIn Key1;
      TranOut Key2;
      pKeyTab tab;
  */
```

Abstract:

Changes the old element into the new one WARNING: THE TABLE MAY NOT BE IN THE SAME ORDER!

Parameters:

OldKey — The old raw information

Key1 — The new raw information

Key2 — The new translated information

tab — Pointer to the keytranslation table to alter

Returns:

success or GR value if failed

6.27.1.3. Function DeleteFromKeyTable

Code:

```
extern GeneralReturn DeleteFromKeyTable ();
  /*  RawIn Key;
      pKeyTab tab;
  */
```

Abstract:

Deletes the element from the table.

Parameters:

Key — The old raw information

tab — Pointer to the keytranslation table to alter

Returns:

success or GR value if failed.

6.27.1.4. Function ConvertToNewVersion

Code:

```
extern GeneralReturn ConvertToNewVersion ();
  /*  char * OldKeyName;
      pKeyTab *pKeyNew;
  */
```

Abstract:

Changes the old version of a keytran file (versions 1-3 which hash using the size of the table, into the current version, which uses a constant since the table may be changed dynamically.

Parameters:

OldKeyName

Absolute pathname of old key tran file

pKeyNew Pointer to new version of table, or Nil if failed

Returns:

Success or GR value if failed. Also returns Success if the file is already the correct version and does not need to be converted.

6.27.1.5. Function SaveKeyTable

Code:

```
extern GeneralReturn SaveKeyTable ();
  /*  pKeyTab tab;
      char * OutFileName;
  */
```

Abstract:

Writes the key map entries to a KEYTRAN file.

Parameters:

Tab Pointer to Keytranslation Table

OutFleName
The absolute name of the file to write the table to.

Returns:

Success or GR value if failed.

6.27.1.6. Function MakeAnEmptyKeyTable

Code:

```
extern GeneralReturn MakeAnEmptyKeyTable ();
  /*  Boolean Raw;
      pKeyTab *pKey;
  */
```

Abstract:

Creates an empty keytranslation table and puts it in memory. Use SaveKeyTable to write it to a file. Essentially it is an empty map table with Table^.rawkeyboard set to TRUE for RawKeyboard values and FALSE for ascii values to be returned. Class is set to standard.

Parameters:

Raw	If TRUE, sets table to be rawkeyboard, else ascii values are returned and nothing for mouse buttons, which is the default.
pKey	Pointer to table, Nil if failed

Returns:

Pointer to table or Nil if failed. Success or GR value if failed.

6.27.1.7. Function DestroyKeyTable

Code:

```
extern GeneralReturn DestroyKeyTable ();
  /*  pKeyTab tab;  */
```

Abstract:

Deallocates the specified key translation table. Do not use it afterwards.

Parameters:

tab The table to deallocate

Returns:

Success or GR value if failed.

6.27.1.8. Function LoadKeyTable

Code:

```
extern GeneralReturn LoadKeyTable ();
  /*  char * name;
      pKeyTab *pKey;
  */
```

Abstract:

Reads in a key translation table from the disk. This file should have been created by the Key Translation Compiler (KeyTranCom) or created using MakeAnEmptyKeyTable and SaveKeyTable or ConvertToNewVersion and SaveKeyTable.

Parameters:

name The absolute name of the file to read the table from. Extension ".KeyTran" may be omitted.

pKey The name of the table in memory

Returns:

Pointer to the table or NIL if not found.

Errors:

If key translation table not found or if mal-formed or if it is the wrong version or if the file does not seem to be a key translation table.

6.27.1.9. Function TranslateKey

Code:

```
extern GeneralReturn TranslateKey ();
  /*  pKeyTab kt;
      EventRec e;
      KeyEvent *k;
  */
```

Abstract:

The main procedure to translate a raw key event from the tracker into an event to be passed to the user.

Parameters:

kt — The key translation table to translate with respect to

e — the raw event from the tracker

k — set to the translated event based on kt

Returns:

Success or GR value if failed

6.27.1.10. Function GetTranslatedEvent

Code:

```
extern KeyEvent GetTranslatedEvent ();
  /*  pKeyTab pKey;
      Window win;
      KeyHowWait howwait;
  */
```

Abstract:

Returns the next keyboard or puck event as a TRANSLATED EVENT. If

howWait is KeyDontWait then returns immediately with a Position or DiffPosition event (or a regular event if one has been queued). If howWait is KeyWaitDiffPos then waits for the x,y position of the cursor to be different. If howWait is KeyWaitEvent then waits for the next key or button transition. If window is not the Listener, then will wait for window to be the Listener before returning unless howWait is KeyDontWait in which case returns the NoEvent event.

Parameters:

pKey — the translation table to use

win — the window that an event is wanted for

howWait — determines how to wait for the event

Returns:

TRANSLATED EVENT record for the event

Errors:

If input has not been enabled for this window

6.27.2. Keytrandefs

Code:

```
#include <SapphFileDefs.h>
#include <SapphLoadFile.h>
#include <SapphDefs.h>
#include <stdio.h>

#define HashmaskNumber 31


/*Keyboard Keys value returned*/

#define KeyHELP 7
```

```
#define KeyBACKSPACE 8
#define KeyTAB 9
#define KeyLF 10
#define KeySETUP 11
#define KeyNOSCROLL 12
#define KeyRETURN 13

#define KeyOOPS 21
#define KeyINS 27
#define KeyESC 27
#define KeySPACE 32

#define KeyDEL 127
#define KeyUPARROW 128
#define KeyBREAK 128
#define KeyDOWNARROW 129
#define KeySHIFTBREAK 129
#define KeyLEFTARROW 130
#define KeyRIGHTARROW 131

#define KeyPF1 132
#define KeyPF2 133
#define KeyPF3 134
#define KeyPF4 139
#define KeyENTER 140
#define KeyNCOMMA 146
#define KeyNMINUS 147
#define KeyNPERIOD 148
#define KeyN0 150
#define KeyN1 151
#define KeyN2 152
#define KeyN3 153
#define KeyN4 154
#define KeyN5 156
#define KeyN6 157
#define KeyN7 158
#define KeyN8 159
#define KeyN9 160
```

```
#define KeyControlSPACE 128 + 32        /* Perq 1 only */
#define KeyControlDEL 128 + 127
#define KeyControlHELP 128 + 7
#define KeyControlBACKSPACE 128 + 8
#define KeyControlTAB 128 + 9
#define KeyControlLF 128 + 10
#define KeyControlOOPS 128 + 21
#define KeyControlRETURN 128 + 13
#define KeyControlESC 128 + 27
#define KeyControlINS 128 + 27
#define KeyControlSETUP 6
#define KeyControlNOSCROLL 14
#define KeyControlUPARROW 28
#define KeyControlDOWNARROW 29
#define KeyControlLEFTARROW 30
#define KeyControlRIGHTARROW 31
#define KeyControlBREAK 130
#define KeyControlSHIFTBREAK 131

#define iYellowMask 001
#define iMiddleMask 001
#define iWhiteMask 002
#define iLeftMask 002
#define iBlueMask 004
#define iGreenMask 010
#define iRightMask 010


DefineString(Key_Name, 255);


/*     Error General Return values    */


#define KeyTran_Error_Base 4600
#define ErElementNotFound KeyTran_Error_Base + 1
#define ErIllegalKeyVersion KeyTran_Error_Base + 2
#define ErNoFreeSpaces KeyTran_Error_Base + 3
#define ErHashZero KeyTran_Error_Base + 4
```

```
#define ErNotValidTable KeyTran_Error_Base + 5


/*------ Exported Constants for key translations ------*/

#define WILDREGION 31      /*no matter what region*/
#define cChCmd 0           /*special reserved command numbers*/
#define cNoCmd 1
#define KeyTranVersion 4   /*Version of the format for the
                               Keytran file*/



/* MAXMAP & WILDREGION must be packable in 14 bits */

#define MAXMAP 511
#define MAPNIL 0



#define Standard 0
#define Control 1
#define Mouse 2
#define NonStandard 3
typedef char KeyKind;



/*keykind tells how complex the event must be before a lookup
  is done in the table.  Mouse means mouse or control.*/


#define CodeNormal 0
      /* return code & change Escape class to 0 */
#define CodeSame 1
      /* return code & leave Escape class unchanged */
#define EscReturn 2
      /* change escape class to 'chval', return cmd */
```

```
#define EscNoop 3
      /* change escape class to 'chval', return NoCmd */
typedef char EscKind;



/* Types of key translation:
  *  1) Standard    -- no explicit table entry exists,
  *                    char code & <CNTRL> used to construct
  *                    'ch', cmd = cChCmd--mouse transitions,
  *                    etc.  ->  cmd = cNoCmd,
  *                    EscClass becomes 0
  *  2) CodeNormal -- find table entry that matches key event,
  *                    escty=CodeNormal
  *                    return cmd, ch  pair from that entry,
  *                    EscClass becomes 0
  *  3) CodeSame    -- returns cmd,ch from matching entry,
  *                    escty=CodeSame,
  *                    EscClass doesn't change
  *  4) EscReturn   -- matching entry has escty=EscReturn,
  *                    return cmd,ch  and
  *                    EscClass becomes ord(ch)
  *  5) EscNoop     -- entry with escty=EscNoop, EscClass becomes
  *                    ord(ch), return cNoCmd
  *  6) GetArg      -- entry with escty=EscNoop, EscClass := ord(ch)
  *                    (probably 0),
  *                    CmdState := cmd, returns cNoCmd
  *                    next key,  returns 'ch' from that entry,
  *                    'cmd' from GetArg
  */


typedef struct
     {
        KeyState hashkey;       /* transition that this entry
                                      describes */
        unsigned int cmd:8;     /* cmd code that it returns */
        unsigned int chval:8;   /* character that it returns;
                                      will be 0..7 if
```

```
                                        escTy = EscReturn or EscNoop */
        unsigned int escty:2;  /* type of entry */
        unsigned int next:9;   /* 0 means NIL */
        unsigned int region:5; /* region for this entry; used
                                  with hashKey and current
                                  region to determine if success */
    } KeyMap;



typedef struct
    {
        unsigned int Version:8;
                 /*format of the key translation table*/
        unsigned int CmdState:8;
        unsigned int RawKeyBoard:1;
                 /* don't translate anything; command comes out
                    cChCmd for all keyboard; for specials,
                    command is 2, ch is special-256 */
        unsigned int EscState:3;
                 /* current Escape sequence state */
        unsigned int NumMaps:9;
                 /* total maps in this keytran table */
        unsigned int Filler:3;
        short HashMask;
        short Len;
                 /* size of total table in bytes */
        KeyKind Class[(MaxCode / 4) + 1];
                 /* 2 bits per event */
        KeyMap Map[1];          /*vble length array*/
    } KeyTransTab;


typedef KeyTransTab *pKeyTab;
```

```
/*      KeyEvent:
          a translated event generated by the user by typing
          on the keyboard or pressing a puck button,
          consisting of a command, char, region, viewport,
          and a pointer position.*/

typedef struct
      {
         Viewport vp;
         short Y, X;
         short region;
         char Cmd;
         char Ch;
      } KeyEvent;


/*  These typedefs were previously in keytran.h  and keytran.c  */

typedef union
              {
                 struct {
                          short Region;
                          unsigned int Code:7;
                                /* key character coding */
                          unsigned int Cntrl0n:1;
                                /* true for keybd chars, if
                                   cntrl key was down */
                          unsigned int Special:1;
                                /* true for transitions not
                                   generated by keybd,
                                   i.e., buttons or regionExit,
                                   etc. */
                          unsigned int EscCl:3;
                               /* escape class - added before
                                  hashing, comes in as zero */
                          unsigned int MButtons:4;
                   } KeyChars;

                /* state of bit pad 1 buttons after event */
```

```
                struct {
                        short reg;
                        short hashks;
                    } RegHash;
            } RawIn;


typedef struct
     {
        short cmd;
        char ch;
        EscKind EscTy;
     } TranOut;


/*   _X_Class is a macro needed to extract two bit fields from
     the Code field of a structure of type KeyState.  x is the
     starting address of the array, and i is the index to the
     two-bit element to be extracted.
*/

#define _X_Class(x,i) (((*(x + (i/4))) >> ((2 * i)%8))
   & (~(-1 << 2)))

/*   _Class is a macro needed to insert two bit fields into the
     Code field of a structure of type KeyState.  x is the
     starting address of the array, i is the index for which
     two bit element is to be inserted, and new is the new value
     to be inserted.
*/

#define _Class(x,i,new) (*(x + (i/4))) =  (((*(x + (i/4)))
   & (~(3 << ((2 * i) % 8)))) \  | (new  << ((2 * i) % 8)) )


GeneralReturn gr;
```

6.28. Malloc

Include Files:

#include <AccentType.h>

6.28.1. Function malloc

Code:

```
extern char * malloc();
 /*  unsigned size;  */
```

Abstract:

Malloc returns a character pointer to "size" bytes of contiguous memory.

Parameters:

size The number of bytes requested

Returns:

A char pointer to the beginning of this size of memory or Null if unable to allocate the memory.

6.28.2. Function free

Code:

```
extern int free();
 /*  char * ptr;  */
```

Abstract:

Free informs the memory system that this process is no longer using this area of memory.

Parameters:

ptr Points to first byte of memory to be freed

Returns:

Indeterminate

6.28.3. Function calloc

Code:

```
extern char * calloc();
  /*  unsigned nelem;
      unsigned elsize;
   */
```

Abstract:

Returns a character pointer to an area of memory large enough to hold nelem elements each elsize bytes big. This memory is cleared.

Parameters:

nelem — Number of elements requested

elsize — Size of each element

Returns:

A character pointer to the beginning of this memory, or Null if unable to allocate.

6.29. ModGetEvent

File ModGetEvent.h is used to allow a client process to receive asynchonous events from more than one window. The code is derived from SapphUser.

Include Files:

```
#include <sapphdefs.h>
```

6.29.1. Function GetEventPort

Code:

```
extern void GetEventPort();
  /*  Window      ServPort;
      KeyHowWait  howWait;
      Port        retPort;
  */
```

Abstract:

Sends a message to the server specified by the "ServPort" parameter, and gives "retPort" as the local Port. The message tells Sapphire that Sapphire should return a message to "retPort" with message ID 2335 when an event occurs. The "howWait" parameter specifies what Sapphire should wait for before it sends a return message. Note that this function would be used in an asynchonous interface with the Sapphire server. The USER must then do a receive on the reply message that is expected from the server ("retPort" is the port on which to receive).

Parameters:

ServPort The Window where the event is expected

howWait What to wait for

KeyWaitDiffPos

Waits for the puck to move

KeyDontWait
Doesn't wait at all

KeyWaitEvent
Waits for either a puck press or a keyboard press

retPort A valid local Port on which to receive the reply message when an event happens.

6.29.2. Function ExtractEvent

Code:

```
extern Boolean ExtractEvent();
  /*  int       *repMsg;
      Viewport *w;
      EventRec *e;
  */
```

Abstract:

Extracts information from a reply message received as a result of having earlier called GetEventPort. The information is then placed in the EventRec given to the function as a parameter. The structure of the message is assumed to be of the following type:

```
typedef struct {
                        Msg       head;
                        TypeType  RetCodeType;
                        short     RetCode;
                        TypeType  IPCNam3;
                        EventRec  Arg3;
                        TypeType  IPCNam4;
                        Port      Arg4;
                        } RepMessage;
```

The "e" parameter gets its value from Arg3 of the message; e->vp gets its

message from Arg4 of the message. The "w" parameter gets its value from the RemotePort specified in the head of the message.

Parameters:

repMsg	A pointer to a message received by the user of the type described above
w	The Window from which the message was generated. (Note that the declared type is "Viewport" but a "Window" is returned.)
e	The EventRec which is to hold the information extracted from the message

Returns:

FALSE	If anything was wrong with the message (i.e. ID!=2335, RetCodeType!= TypeInt16, RetCode!=Success, IPCNam3!= TypeUnstructured, or IPCNam4!=TypePt)
TRUE	If message appeared to be successfully received and interpreted and the parameters were successfully unpacked