IBM RT PC Advanced Interactive Operating System Version 2.1

AIX Operating System Technical Reference

Volume 2

Programming Family

IBM

Personal
Computer
Software

SC23-0809-0

IBM RT PC Advanced Interactive Operating System Version 2.1

AIX Operating System Technical Reference

Volume 2

Programming Family

IBM

Personal
Computer
Software

SC23-0809-0

First Edition (January 1987)

Portions of the code and documentation described in this book were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

This edition applies to Version 2.1 of the Advanced Interactive Executive Operating System, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

International Business Machines Corporation provides this manual "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this manual at any time.

Products are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM RT PC dealer or your IBM marketing representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758-3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

IBM TECHNICAL NEWSLETTER

for the

RT Personal Computer

AIX Operating System Technical Reference
Volume 2

—OVER—

Order Numbers:
08F3407
SN20-9881
September 25, 1987

TB 08F3407
Printed in U.S.A.

Summary of Changes

This technical newsletter contains updates to the Version 2.1.1 publication to include changes made for Version 2.1.2. File the Version 2.1.1 TNL (SN20-9869) first, then apply the enclosed update to those pages.

A vertical bar in the left margin of this TNL indicates material that has been updated or added for Version 2.1.2.

Perform the following:

Remove Pages	**Insert Update Pages**
iii and iv	iii and iv
5-59 to 5-68	5-58.1 to 5-68.6
None	Chapter 8
None	A-9 to A-12
Index	Index

Note: Please file this cover letter at the back of the manual to provide a record of changes.

September 25, 1987

IBM TECHNICAL NEWSLETTER

for the

RT Personal Computer

AIX Operating System Technical Reference
Volume 2

—OVER—

Order Numbers:
92X1321
SN20-9869
June 26, 1987
© Copyright IBM Corp. 1987

Summary of Changes

This technical newsletter contains updates to the Version 2.1 publication to include changes made for Version 2.1.1.

Perform the following:

Remove Pages	**Insert Update Pages**
4-43 and 4-44	4-43 and 4-44
4-53 to 4-56	4-53 to 4-56
4-65 to 4-74	4-65 to 4-74
4-93 to 4-112	4-93 to 4-112
4-123 and 4-124	4-123 and 4-124
4-139 and 4-140	4-139 and 4-140
4-153 to 4-158	4-153 to 4-158
5-55 to 5-58	5-55 to 5-58
6-11 to 6-14	6-11 to 6-14
6-121 and 6-122	6-121 and 6-122
All of Chapter 7	All of Chapter 7
A-7 and A-8	A-7 and A-8
C-23 and C-24	C-23 to C-24.2
The Index	The Index

Note: Please file this cover letter at the back of the manual to provide a record of changes.

June 26, 1987

Contents

Volume 1. System Calls and Subroutines

Volume 2. Files and Device Drivers

Figures

Volume 2. Files and Device Drivers

Chapter 4. File Formats

About This Chapter

This chapter outlines the formats of various files. The C language **struct** declarations for the file formats are given where applicable. These structures are usually found in header files located in the **/usr/include** or **/usr/include/sys** directories, although they can be located in any directory in the file system.

Many of the files described in this chapter contain ***magic numbers*** at predefined offsets. Magic numbers provide programs with a way to verify the format of an input file before attempting to process it. The values used for magic numbers are chosen because they are not likely to occur as a random pattern in normal input.

.init.state

Purpose

Specifies the initial state for the AIX Operating System.

Description

The **/etc/.init.state** file specifies the initial state in which the **init** process is to start up the AIX Operating System. This file contains a single line that specifies one of the following initial states:

m Maintenance mode (single-user mode). When maintenance mode is entered, no devices have been configured, no file systems have been mounted, and no daemon processess have been started. These operations are normally performed by the **/etc/rc** shell procedure, which is not run when entering maintenance mode. See the **/etc/rc** file on your system for the commands that perform these operations.

a Automatic multi-user mode. Enters multi-user mode, passing an `a` to **/etc/rc** as the first parameter, **$1**.

c "Clean" multi-user mode. Enters multi-user mode, passing a `c` to **/etc/rc** as the first parameter, **$1**. By convention, the `c` indicates that the file systems are probably in good condition, or "clean." You can customize the **/etc/rc** file on your system to skip running **fsck** in order to shorten the system-startup procedure. Note, however, that "clean" mode does *not* guarantee that any file systems are in good condition.

d "Dirty" multi-user mode. Enters multi-user mode, passing a `d` to **/etc/rc** as the first parameter, **$1**. By convention, the `d` indicates that one or more file systems may have been damaged. **/etc/rc** should run the **fsck** command to check all file systems.

e *file* Exec mode. Executes the shell procedure named *file*. When the shell procedure terminates, the system asks the operator whether to enter maintenance mode or multi-user mode.

o Operator mode. Asks the operator whether to enter maintenance mode or multi-user mode. The system waits until a response is given.

u Unknown mode. Asks the operator whether to enter maintenance mode or multi-user mode. If no response is given within a period of time (approximately a minute), the system enters automatic multi-user mode.

If the operator selects multi-user mode in response to a prompt, then **init** asks whether to check the file systems. The response to this question determines whether a `c` or a `d` is passed to **/etc/rc**.

The **/etc/.init.state** file can also contain comment lines, which are indicated by a # character in the first column.

File

/etc/.init.state

Related Information

In this book: "Creation and Execution" on page 1-16, "iplvm, waitvm" on page 2-58, and "reboot" on page 2-109.

The **init** and **rc** commands in *AIX Operating System Commands Reference.*

a.out

Purpose

Provides common assembler and link editor output.

Synopsis

#include <a.out.h>

Description

The **as** (assembler) and **ld** (link editor) programs produce an output file (the **a.out** file by default) in the following format. The **a.out** file is executable if the assembler and the link editor do not find any unresolved external references or errors in the source.

This file can have the following sections: a header, the text segment, data segment, relocation information, a symbol table, a line number section, a string table, and a shared library identifer (in that order). The last five sections may be missing if the program was linked with the **-s** flag of the **ld** command or if they were removed by the **strip** command. The shared library identifier exists only for object modules related to a shared library image. Note the relocation information is not present if there are not external references to be resolved after linking.

Loading an **a.out** file into memory for execution causes the creation of three logical segments: the text segment, the data segment (initialized data followed by data that is not initialized, the latter actually being initialized to all zeros) and a stack.

Segment 1 occupies a low memory address in the process image and its size is static. Segment 2 follows segment 1 in memory. The size of this segment can be extended using the **brk** system call. The stack segment begins near the highest locations in segment 3 and grows toward segment 2 as required.

Header

The format of the **a.out** header is:

```
struct exec  {
    unsigned char a_magic[2]; /* magic number */
    unsigned char a_flags;    /* flags */
    unsigned char a_cpu;      /* CPU-ID */
    unsigned char a_hdrlen;   /* length of header */
    unsigned char a_unused;   /* reserved for future use */
```

```
        unsigned short a_version; /* version stamp */
        long a_text;              /* size of text segment */
        long a_data;              /* size of data segment */
        long a_bss;               /* size of bss segment */
        long a_entry;             /* entry point */
        long a_misc;              /* misc., e.g. initial stack pointer */
        long a_syms;              /* symbol table size */
        /* SHORT FORM ENDS HERE */
        long a_trsize;            /* text relocation size */
        long a_drsize;            /* data relocation size */
        long a_tbase;             /* text relocation base */
        long a_dbase;             /* data relocation base */
        long a_lnums;             /* size of line number section */
        long a_toffs;             /* offset of text from start of file */
    };
```

The fields in the header are as follows:

a_magic A 2-byte number that has a value of 0x0103.

a_flags A byte with various options that apply to the **a.out** file. Bits that are not used are set to 0. Options supported are:

A_TOFF	Text offset is specified by **a_toffs**
A_STRS	String table is present
A_HDREXT	Extended header is present
A_EXEC	File is executable
A_SEP	Instruction and data spaces are separate
A_PURE	Pure text
A_SHLIB	Shared library identifier is present

a_cpu A coded entry describing the system unit and the byte order it expects. The coded entry for RT PC is 0x13.

a_hdrlen The length of the header. The size of the header is variable, but it must be at least 32 bytes to include all of the fields in the structure through **a_syms**. If the size of the header is such that a field is not included, the default value is assumed.

a_misc The maximum size in bytes the user stack is allowed to grow.

Extended Header

The presence of an extended header is indicated by the A_HDREXT bit being set in **a_flags**. The format of the extended header is:

```
struct exthdr  {
    unsigned short ax_size;     /* total size of extension */
    unsigned short ax_type;     /* type of extension */
    unsigned short ax_flags;    /* e.g., execution model */
    unsigned short ax_nsegs;    /* number of segment entries */
};
```

The size of the extension (in bytes) is **ax_size**, which includes the length of **exthdr** plus any auxiliary entries which comprise this extended header type, indicated by **ax_type**. The value of **ax_flags** is also dependent on **ax_type**. In the event that the following auxiliary entries contain per-segment information, **ax_nsegs** is the number of segments (and thus the number of auxiliary entries) present.

Legal values for **ax_type** are:

AXT_INTEL	1
AXT_SHLIB	2

The legal values for **ax_flags** when **ax_type** is AX_INTEL are:

AXF_SSS	Separate stack segment
AXF_MCS	Multiple code segments
AXF_MDS	Multiple data segments
AXF_HDS	Huge data present
AXF_OVLY	Code overlay
AXF_FPH	Floating-point hardware required
AXF_ABS	Absolute addresses present

When **ax_type** is AXT_INTEL, **exthdr** is followed by **ax_nsegs** entries of the form:

```
struct segent  {
    unsigned short as_type;     /* segment type */
    unsigned short as_flags;    /* segment attributes */
    unsigned short as_num;      /* segment number */
    unsigned short as_nlnno;    /* # lineno entries */
    long as_filep;              /* position (offset) in file */
    long as_psize;              /* size of segment in file */
    long as_vsize;              /* virtual size */
    long as_rsvd1;              /* reserved */
    long as_rsvd2;              /* reserved */
    long as_lnptr;              /* position of lineno entries */
```

```
   };
```

Each **segent** describes a segment of the **a.out** file. Legal values for the type of segment, **as_type**, are:

AST_NULL	
AST_TEXT	Code segment
AST_DATA	Data segment

Various characteristics of the segment are described by **as_flags**. Possible values are:

ASF_HUGE	Segment contains huge model data
ASF_BSS	Segment contains implicit bss
ASF_SHARE	Segment is sharable
ASF_EXPDOWN	Segment expands downward
ASF_SEG	Always on for segments

When **ax_type** is AXT_SHLIB, **exthdr** is followed by a table describing the **ax_nsegs** shared libraries required by this program. Each element of the table has the format:

```
   struct slent  {
       long sl_off;        /* offset from table start of lib key */
       long as_addr;       /* address where library to be mapped */
   };
```

The table is terminated by an element with an **sl_off** member of zero. Following the table are the shared library keys associated with the libraries mentioned. Each shared library key is preceded by a string recognizable to the **what** command, and is terminated with an ASCII NUL character. (Each **sl_off** entry points past the **what** string to the real start of the key.)

Text and Data Sections

The text and data sections are indicated in the fields as follows:

a_text The size of the text segment in bytes. This segment begins immediately after the header or at the offset specified in the **a_toffs** field if the **A_TOFF** flag is set. The **A_TEXTPOS** macro defined in the **a.out.h** header file gives the offset of this segment in either case.

a_data The size of the data segment in bytes. This segment begins immediately following the text segment. The A_DATAPOS macro gives the offset of this segment.

a_bss The size of the bss segment in bytes. This segment represents data that is not initialized. It does not appear in the file.

The text, data, and bss segments must each be a multiple of full words in size.

a_entry The text address where the program should start to run. The default is the *a_tbase* value.

a_tbase The virtual address of the first byte of the text segment. The default value for this field is 0.

a_dbase The virtual address of the first byte of the data segment. The default value for this field is **a_tbase** + **a_text**, rounded to the next segment boundary.

Relocation

The fields in the relocation information are as follows:

a_drsize The size of the data relocation information in bytes. The A_DRELPOS macro defines where the data relocation information entries begin.

a_trsize The size of the text relocation information in bytes. The A_TRELPOS macro defines where the text relocation entries begin.

A word in the text or data segment of memory contains either an actual value or the value of an offset. If a word in the text or data segment references an undefined external symbol, its value is an offset from the associated external symbol. During processing, the link editor defines the external symbol and adds the value of the symbol to the word in the file.

When relocation information is present, each item that can be relocated is 8 bytes long. The format of the relocation information is:

```
struct reloc {

   long r_vaddr;            /* virtual address of reference */
   unsigned short r_symndx;  /* internal segnum or extern
                               symbol number */
   unsigned short r_type;    /* relocation type */
};
```

The **r_vaddr** field gives the location of the relocatable reference relative to the beginning of the segment in which it is defined.

The **r_symndx** field contains a symbol number in the case of an external. Otherwise, it contains a segment number code:

S_ABS	0xFFFF	/* absolute */
S_TEXT	0xFFFE	/* text segment */
S_DATA	0xFFFD	/* data segment */
S_BSS	0xFFFC	/* bss segment */

The **r_type** field indicates the type of relocation. The relocation types are:

R_ABS	0	/* absolute */
R_RELBYTE	2	/* byte */
R_PCRBYTE	3	/* byte (pc relative) */
R_RELWORD	4	/* word */
R_PCRWORD	5	/* word (pc relative) */
R_RELLONG	6	/* long */
R_PCRLONG	7	/* long (pc relative) */
R_REL3BYTE	8	/* 3 bytes */
R_KBRANCH	9	/* 20-bit 1-shifted */
R_SEG86	10	/* segmented PC-XT */
R_SEG286	11	/* segmented PC-AT */
R_KCALL	12	/* 20-bit 1-shifted or fix up */

Symbol Table

The **a_syms** field in the header indicates the size of the symbol table in bytes. The A_SYMPOS macro defines the offset where the symbol table begins.

The symbol table consists of the following entries:

```
struct syment  {
   union {
       char  _n_name [8];      /* non-flex version */
     struct  {
           long  _n_zeroes;    /* flexname == 0 */
           long  _n_offset;    /* offset into string table */
           }_n_n;
       char   *_n_n_ptr[2];    /* allows for overlaying */
   }_n;
   long n_value;               /* symbol value */
   unsigned char   n_sclass;  /* storage class */
   unsigned char   n_numaux;  /* number of auxiliary entries    */
   unsigned short  n_type;    /* language base and derived type */
};

#define SYMENT struct syment
#define SYMESZ sizeof(struct syment)
```

```
#define n_name _n._n_name
#define n_nptr _n._n_nptr[1]
#define n_zeroes _n._n_n._n_zeroes
#define n_offset _n_n._n_n._n_offset
```

The low-order 3 bits of **n_sclass** indicate the section information:

N_UNDF	00	/* undefined */
N_ABS	01	/* absolute */
N_TEXT	02	/* text */
N_DATA	03	/* data */
N_BSS	04	/* bss */
N_COMM	05	/* common */
N_SECT	07	/* section mask */

The high-order bits indicate the storage class. The following storage classes are implemented:

C_NULL	0000	/* undefined symbol */
C_AUTO	0010	/* (0x08) automatic variable */
C_EXT	0020	/* (0x010) external symbol */
C_STAT	0030	/* (0x18) static */
C_REG	0040	/* (0x20) register variable */
C_EXTDEF	0050	/* (0x28) external definition */
C_LABEL	0060	/* (0x30) label */
C_ULABEL	0070	/* (0x38) undefined label */
C_MOS	0100	/* (0x40) member of structure */
C_ARG	0110	/* (0x48) function argument */
C_STRTAG	0120	/* (0x50) structure tag */
C_MOU	0130	/* (0x58) member of union */
C_UNTAG	0140	/* (0x60) union tag */
C_TPDEF	0150	/* (0x68) type definition */
C_USTATIC	0160	/* (0x70) undefined static */
C_ENTAG	0170	/* (0x78) enumeration tag */
C_MOE	0200	/* (0x80) member of enumeration */
C_REGPARM	0210	/* (0x88) register parameter */
C_FIELD	0220	/* (0x90) bit field */
C_BLOCK	0300	/* (0xc0) **.bb** or **.eb** */
C_FCN	0310	/* (0xc8) **.bf** or **.ef** */
C_EOS	0320	/* (0xd0) end of structure */
C_FILE	0330	/* (0xd8) file name */
N_CLASS	0370	/* (0xff) storage class mask */

If a symbol section and class is **undefined external** and the value field is a value other than 0, the link editor interprets the symbol as the name of a common region in which the size is indicated by the value of the symbol.

The **n_type** field is primarily for use by a symbol debugger. The low-order 4 bits form the base type with values defined as follows:

T_NULL	0	/* undefined symbol */
T_ARG	1	/* used internally by compiler */
T_CHAR	2	/* character */
T_SHORT	3	/* short integer */
T_INT	4	/* integer */
T_LONG	5	/* long integer */
T_FLOAT	6	/* floating point */
T_DOUBLE	7	/* double */
T_STRUCT	8	/* structure */
T_UNION	9	/* union */
T_ENUM	10	/* enumeration */
T_MOE	11	/* member of enumeration */
T_UCHAR	12	/* unsigned character */
T_USHORT	13	/* unsigned short */
T_UINT	14	/* unsigned integer */
T_ULONG	15	/* unsigned long */

The high-order bits form the derived type. The following values are repeated up to six times to form the derived type:

DT_NON	0	/* no derived type */
DT_PTR	1	/* pointer */
DT_FCN	2	/* function */
DT_ARY	3	/* array */

The **n_numaux** field contains the number of auxiliary entries associated with this symbol table entry. Currently, a symbol table entry can have at most one auxiliary entry. The auxiliary entry provides additional information, and has this form:

```
union auxent {
  struct {
     long x_tagndx;                /* str, union, or enum tag index */
     union {
       struct  {
          ushort x_lnno;           /* declaration line number */
          ushort x_size;           /* str, union, array size */
      }x_lnsz;
      long x_fsize;                /* size of function */
   } x_misc;
```

```
         union {
            struct {                        /* if ISFCN, tag, or .bb */
               long x_lnnoptr;              /* ptr to fcn line # */
               long x_endndx;               /* entry index past block end */
            } x_fcn;
            struct {                        /* if ISARY, up to 4 dimen. */
               ushort x_dimen[DIMNUM];
            } x_ary;
         }x_fcnary;
       }x_sym;
       struct {
          char x_fname[FILNMLEN];
       }x_file;
      };

      #define FILNMLEN    14
      #define DIMNUM      4
```

The information in an auxiliary entry cannot be correctly interpreted without the symbol table entry to which it belongs. The order of entries within the symbol table is significant.

Line Number Section

The **a_lnums** field contains the size in bytes of the line number section. The line number section starts at the location in the file defined by the A_LINEPOS macro.

Line number entries are used by the symbolic debugger to debug code at the source level. Entries within the line number section are grouped by function. The format of a line number entry is:

```
   struct lineno {
      union {
        long l_symndx;          /* symbol table index of function name
                                   if and only if l_lnno == 0 */
        long l_paddr;           /* physical address of line number */
     } l_addr;
     unsigned short l_lnno;     /* line number */
   };
```

String Table

The string table contains the names of symbols that are longer than 8 characters. It is present only if the A_STRS flag is set. If present, the first 4 bytes contain the length, in bytes, of the string table, including the count. The remainder of the table is a sequence of null-terminated strings. If the **n_zeroes** field in a symbol entry is 0, the **n_offset** field gives the offset into the string table of the name for the symbol.

Shared Library Identifier

The shared library identifier names the shared library image to which this object module is related. It is present only if the A_SHLIB flag is set. If present, the first byte contains the length of the identifier section including the count byte. The identifier itself is a string terminated with an ASCII NUL.

Related Information

The **as, cc, dump, ld, nm, sdb, size,** and **strip** commands in *AIX Operating System Commands Reference.*

The **config** program and the **what** command in *AIX Operating System Commands Reference.*

acct

Purpose

Provides the accounting file format for each process.

Synopsis

#include <sys/acct.h>

Description

The accounting files provide a means to monitor the use of the system. These files also serve as a method for billing each process for processor usage, materials, and services. The **acct** system call produces accounting files. The **<sys/acct.h>** file defines the records in these files. The content of the records are:

```
/* Accounting structures */
typedef ushort comp_t;     /* floating point */
           /* 13-bit fraction, 3-bit exponent */

struct acct
{
     char  ac_flag;    /* Accounting flag */
     char  ac_stat;    /* Exit status */
     ushort  ac_uid;   /* Accounting user-ID */
     ushort  ac_gid;   /* Accounting group-ID */
     dev_t  ac_tty;    /* control typewriter */
     time_t  ac_btime; /* Beginning time */
     comp_t  ac_utime; /* accounting user time in clock ticks */
     comp_t  ac_stime; /* accounting system time in clock ticks */
     comp_t  ac_etime; /* accounting elapsed time in clock ticks */
     comp_t  ac_mem;   /* memory usage */
     comp_t  ac_io;    /* chars transferred */
     comp_t  ac_rw;    /* blocks read or written */
     char  ac_comm[8]; /* command name */
};
```

```
extern struct  acct  acctbuf;
extern struct  inode  *acctp; /* i-node of accounting file */

#define AFORK  01             /* has executed fork, but no exec */
#define ASU    02             /* used superuser authority */
#define ACCTF  0300           /* record type: 00 = acct */
```

The fields are as follows:

ac_comm This field contains the command name. A child process, created by a **fork** system call, receives this information from the parent process. An **exec** system call resets this field.

ac_flag This field indicates whether the process used superuser authority, or it was created using a **fork** command but not yet followed by an **exec** system call. The **fork** command turns the AFORK flag in this field on and the **exec** system call turns the AFORK flag off.

ac_mem This field contains memory usage. For each clock tick, the system updates this field with the current process size and charges usage time to the process. This is computed as ((*data size*) + (*text size*)) ÷ (*number of in-memory processes using text*)

The following structure (not part of **acct.h**) represents the total accounting format used by the various accounting commands:

```
/* Float arrays below contain prime time and non-prime time
   components */

   struct tacct {
     uid_t  ta_uid;            /* user-ID */
     char   ta_name[8];        /* login name */
     float  ta_cpu[2];         /* cum. CPU time, p/np (mins) */
     float  ta_kcore[2];       /* cum. kcore-mins, p/np */
     float  ta_io[2];          /* cum. chars xferred (512s) */
     float  ta_rw[2];          /* cum. blocks read/written */
     float  ta_con[2];         /* cum. connect time, p/np, mins */
     float  ta_du;             /* cum. disk usage */
     long   ta_qsys;           /* queuing sys charges (pgs) */
     float  ta_fee;            /* fee for special services */
     long   ta_pc;             /* count of processes */
     unsigned short  ta_sc;    /* count of login sessions */
     unsigned short  ta_dc;    /* count of disk samples */
   };
```

File

/usr/include/sys/acct.h

Related Information

In this book: "acct" on page 2-11 and "utmp, wtmp, .ilog" on page 4-170.

The **acctcom** command in *AIX Operating System Commands Reference.*

The **acct, acctcms, acctcon, acctmerg, acctprc, acctsh, diskusg,** and **runacct** procedures in *AIX Operating System Commands Reference.*

ar

Purpose

Describes common archive file format.

Synopsis

#include <ar.h>

Description

The **ar** (archive) command is used to combine several files into one. The **ar** command creates an **ar** file. The **ld** (link editor) searches archive files to resolve program linkage.

Each archive begins with the archive magic string:

```
#define ARMAG  "!<arch>\n"   /* magic string */
#define SARMAG  8           /* length of magic string */
```

Each archive that contains common object files includes an archive symbol table. See "a.out" on page 4-5 for the format of an object file. **ld** uses this symbol table to determine the archive members to load during the link edit process. The archive symbol table, if it exists, is always the first file in the archive. It is never listed, but **ar** automatically creates and updates it.

The archive file members follow the archive header and symbol table. A file member follows each file member header. The format of a file member header is:

```
#define ARFMAG    "\n"      /* header trailer string */

struct ar_hdr {       /* file member header */
  char  ar_name[16];  /* file member name - terminated by '/'*/
  char  ar_date[12];  /* file member date */
  char  ar_uid[6];    /* file member user identification */
  char  ar_gid[6];    /* file member group identification */
  char  ar_mode[8];   /* file member mode */
  char  ar_size[10];  /* file member size */
  char  ar_fmag[2];   /* ARFMAG - string to end header */
};
```

All information in the file member header is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers, except **ar_mode**, which is stored in octal. Thus, if the archive contains printable files, you can print the archive.

The **ar_name** field is blank-padded and terminated by a / (slash). The **ar_date** field indicates the date the file was last modified prior to archive. The **ar** command allows archives to move from system to system.

Each archive file member begins on an even-byte boundary. **ar** inserts null bytes for padding and a new-line character between files, if necessary. The **ar_size** field is the size of the file without padding. An archive file contains no empty areas.

If the archive symbol table exists, the first file in the archive has a zero-length name, for example, **ar_name[0]** = = '/'. The contents of the symbol table are as follows:

> The number of symbols. This is 4 bytes long.
>
> The array of offsets into the archive file. The length is determined by 4 bytes times the number of symbols.
>
> The name string table. The size is determined by **ar_size** minus (4 bytes times (the number of symbols plus 1)).

The **sgetl** and **sputl** functions manage the number of symbols and the array of offsets. The string table contains an equal number of null-terminated strings and elements in the offsets array. Each offset from the array associates with the corresponding name from the string table, in order. The string table names all the defined global symbols found in the object files contained in the archive. Each offset locates the archive header for the associated symbol.

Related Information

In this book: "sgetl, sputl" on page 3-334 and "a.out" on page 4-5.

The **ar**, **ld**, and **strip** commands in *AIX Operating System Commands Reference.*

attributes

Purpose

Describes an attribute file format.

Description

ASCII files are used to control some RT PC utilities in order to simplify installing, customizing, and maintaining RT PC. A text editor can be used to examine or change these files. These files share an attribute-structured format.

An attribute-structured file consists of one or more named stanzas, separated by blank lines. Each stanza begins with a name followed by a colon, and contains assignments to keyword **attributes**. The values assigned can be alphanumeric strings or arbitrary character strings enclosed in double quotes. The assignments can associate either a single value or a succession of values with each **attribute**.

Typically, the stanza name is the name of a device or service. The attributes describe the properties or handling of the named device or service. The meanings of the stanza names, **attribute** names, and values are specific to an application. Examples of this file type distributed with the system are **/etc/filesystems**, **/etc/ports**, and **/etc/qconfig**.

The stanza name **default** can be used to specify default values for any **attributes**. These default assignments are implicitly included in every stanza that follows. A specified value overrides the default value. A new **default** stanza automatically cancels all previously specified default values. The syntax of a file of this type is:

```
<file>          ::=  <stanza>
                ::=  <file> <blank line> <stanza>
<stanza>        ::=  <name>:
                ::=  <name>:<assignments>
<name>          ::=  file name or information similar in syntax
<assignments>   ::=  <assignment>
                ::=  <assignments> <assignment>
<assignment>    ::=  <attribute>=<values>
<attribute>     ::=  string of alphanumeric characters
<values>        ::=  <value>
                ::=  <values>,<value>
<value>         ::=  string of alphanumeric characters
                ::=  "arbitrary characters"
```

Lines beginning with an * (asterisk) are considered to be comments and are ignored.

Note: Make sure that the values assigned to attribute keywords do not contain blanks unless they are enclosed in double quotes.

Related Information

In this book: “cc.cfg” on page 4-29, “connect.con” on page 4-33, “filesystems” on page 4-64, “master” on page 4-98, “ports” on page 4-117, “qconfig” on page 4-129, “rasconf” on page 4-133, and “system” on page 4-139.

autolog

Purpose

Performs login function automatically.

Description

The optional **autolog** file causes the RT PC system to perform a login sequence automatically when it contains a valid user name. When power is applied to the system and the login port is the console, **login** searches for this file. If this file is found, **login** creates a session for a specific user automatically. The **autolog** file is an ASCII file containing a valid user name. A user can create this file while customizing the system. After it is created, this file can be edited using any editor. If this file does not exist, **login** causes the user to login as usual.

File

/etc/autolog

Related Information

The **login** command in *AIX Operating System Commands Reference.*

backup

Purpose

Copies file system onto temporary storage media.

Synopsis

#include <backup.h>

Description

A backup of the file system provides protection against substantial data loss due to accidents or error. The **backup** command writes file system backups and conversely, the **restore** command reads file system backups. The following text describes the format of a file system backup.

Header Types

The backup contains several different types of header records along with the data in each file that is backed up. The type of header records are:

FS_VOLUME The volume label. This header exists on every volume.

FS_FINDEX An index of files on this volume. Multiple headers of this type can appear on a volume if there are too many i-nodes for the initial index. This header is followed by data.

FS_CLRI A bit map of i-nodes on the file system. A zero bit indicates the i-node is not in use. This header exists only on the first volume. If the backup is a level-zero backup, this header is omitted.

FS_BITS Another bit map of i-nodes. A one bit indicates the i-node is present on this volume or a subsequent volume. This header may not appear on all volumes.

FS_VOLEND Indicates the end of the current volume. This header may not appear on all volumes. This header is used to indicate that all index entries on this volume are used.

FS_END Indicates the end of the backup. This header appears on every volume.

FS_INODE Describes a single i-node. This header is followed by data that consists of directories then followed by the other files within the directories.

FS_NAME A description of a file that is backed up by name.

Header Sequence

The header sequence varies depending on whether the files are backed up by i-node or by name and on the type of backup device used.

Volume 1 of i-node backups to direct access volumes have the following sequence, assuming that more than one volume is required for backup:

FS_VOLUME
FS_CLRI
FS_BITS
FS_FINDEX, followed by data
FS_FINDEX (if applicable), followed by data
FS_END

Subsequent volumes have the following sequence:

FS_VOLUME, followed by data
FS_FINDEX, followed by data
FS_FINDEX (if applicable), followed by data
FS_END

I-node backups to tapes have the same format as previously described, except there are no **FS_FINDEX** headers and the **FS_BITS** header appears on every volume.

The format of backups by name does not depend on the output device. These backups have a simple format:

FS_VOLUME Appears on each volume.

FS_NAME Precedes the data for each file. The files are copied in the order they were named.

FS_END Concludes the backup.

Header Format

The location and size of the headers are independent of any blocking for either the file system or the backup device. Each header begins on an 8-byte boundary. The length of a header depends on its type, but is always padded to a multiple of 8 bytes. Data from a file is similarly padded. Some headers contain addresses of other headers that are the offset in 8-byte units from the beginning of the backup volume.

Each field in a header is written in low-order bytes first for portability. I-node numbers within directories also follow this order. The header begins with the following structure:

```
struct hdr {
       unsigned char  len;
       unsigned char  type;
       ushort         magic;
       ushort         checksum;
};
```

The fields in this header indicate the following information:

len The length of the header in 8-byte units.

type The type of the header.

magic The magic number, which identifies this file as a file system backup. The magic number is one of the following values:

- **MAGIC** Identifies this as a regular file system backup.
- **PACKED_MAGIC** Identifies this as a packed, or compressed, file system backup. Each data file within it is compressed using the same algorithm that is used by the **pack** command. Header information is not compressed.

checksum A checksum.

Volume Headers

FS_VOLUME headers have the following structure:

```
struct {
      struct hdr h;
      ushort     volnum;
      time_t     date;
      time_t     budate;
      daddr_t    numwds;
      char       disk[16];
      char       fsname[16];
      char       user[16];
      short      incno;
 };
```

The fields contain the following information:

volnum Contains the volume number.

date Indicates the date the backup was made.

budate Indicates that all files changed since this date are backed up.

numwds Indicates the number of 8-byte words in this backup.

disk Identifies the device that was backed up.

fsname Identifies the logical name of the backed-up device, for example, /a.

user Identifies the user that made the backup.

incno Shows the level number of the backup.

For backups by name, **budate**, **disk**, and **fsname** have no meaning, and **incno** is 100.

Index Headers

FS_FINDEX records are as follows:

```
struct {
    struct hdr h;
    ushort     dummy;
    ino_t      ino[80];
    daddr_t    addr[80];
    daddr_t    link;
 };
```

The fields are:

ino I-numbers of files indexed

addr Addresses of file indexed

link Address of next index on this volume, or 0 if this is the last.

Bit Maps

FS_CLRI and **FS_BITS** headers have the same structure:

```
struct {
   struct hdr h;
   ushort     nwds;
};
```

In both cases, the bit map follows the header, and **nwds** gives the length of the map in 8-byte units. To save space, some zero bits at the end of the map are not backed up.

File Headers

FS_INODE and **FS_NAME** headers have similar formats:

```
struct {
   struct hdr h;
   ushort    ino;
   ushort    mode;
   ushort    nlink;
   ushort    uid;
   ushort    gid;
   off_t     size;
   time_t    atime;
   time_t    mtime;
   time_t    ctime;
   ushort    devmaj;
   ushort    devmin;
   ushort    rdevmaj;
   ushort    rdevmin;
   off_t     dsize;
   char      name[4];
};
```

The fields **mode** through **ctime** are copied from the i-node on disk.

Other fields are:

ino I-number of file.

devmaj Major device number of file system containing this file.

devmin Minor device number of file system containing this file.

rdevmaj Major device number of this file (character- and block-special files only).

rdevmin Minor device number of this file (character- and block-special files only).

dsize Size of the file after backup. This differs from **size** if the file was compressed during backup.

name The null-terminated name of the file that is supplied by the user. This field is absent from **FS_INODE** headers.

End of Volume or Backup

FS_VOLEND and **FS_END** headers contain only the **hdr** structure.

Backup History

A backup history is kept in the **/etc/budate** file. The entries are in no particular order. Each entry has the following format:

```
struct {
     char    id_name[16];
     char    id_incno;
     time_t  id_budate;
};
```

The fields of each entry are:

id_name	Name of the file system
id_incno	Incremental level number (0-9)
id_budate	Date of most recent backup of the file system at that level.

Related Information

In this book: "filesystems" on page 4-64.

The **backup**, **pack**, and **restore** commands in *AIX Operating System Commands Reference.*

cc.cfg

Purpose

Defines values used by the C compiler.

Description

The **cc.cfg** file defines values used by the **cc** program to run compilers. Normally, the **cc.cfg** file contains entries only for the C compiler provided with the system. Entries are made to this file to support C compilers for other systems as they are added.

This file is an attribute file. The name you specify when you run the **cc** program (it can be linked to several difference names) determines which stanza of the **cc.cfg** file is used. Normally, the **cc** program runs as **cc**; therefore, the first stanza is almost always selected. If the **scc** program (standalone C compiler) is run, then the **scc** stanza is selected. If the **fcc** program (floating point) is selected, then the **fcc** stanza is selected. If the **vcc** program (**a.out**-to-**toc** conversion) is selected, then the **vcc** stanza is selected.

You can specify the following attributes:

as The path name to be used for the assembler.

asflags A string of values, separated by commas, to be passed to the assembler.

asopt A string naming optional flags that, if encountered on the **cc** command line, should be passed to the assembler. See description of the **cppopt** field.

ccom The path name to be used for the compiler. For a one-program compiler, this is the only compiler program provided. For a two-program compiler, this is the parser for the front end (also known as **c0**).

ccomflags A string of values, separated by commas, to be passed to the compiler.

ccomopt A string naming optional flags that, if encountered on the **cc** command line, should be passed to the compiler. See **oppopt**.

cgen The path name to be used for the code generator of a two-program compiler (also known as **c1**).

cgenflags A string of values, separated by commas, to be passed to the code generator. If a one-program compiler is used, these are appended to **ccomflags**.

cgenopt A string naming optional flags that, if encountered on the **cc** commands line, should be passed to the code generator. See **cppopt**.

copt The path name to be used for the peephole optimizer of a compiler with an explicit peephole program (also known as **c2**).

coptflags A string of values, separated by commas, to be passed to the peephole optimizer.

coptopt A string naming optional flags that, if encountered on the **cc** command line, should be passed to the peephole optimizer. See **cppopt**.

cpp The path name to be used for the preprocessor.

cppflags A string of flags, separated by commas, to be passed to the preprocessor.

cppopt A string naming optional flags that, if encountered on the **cc** command line, should be passed to the preprocessor. The string is formatted for **getopt()** subroutine, as a concatenation of flag letters, with a letter followed by a : (colon) if the corresponding flag takes a parameter.

crt, mcrt The path name of the object file passed as the first parameter to the link editor. In the presence of the **−p** flag to **cc**, the **mcrt** value is used; otherwise the **crt** value is used. The defaults are **/lib/crt0.o** and **/lib/mcrt0.o**.

csuffix The suffix for **C** source programs, default **c**.

hsuffix A second suffix for **C** source (enabled by using the **−h** flag to the **cc** command), default **h**.

ld The path name to be used for the link editor.

ldflags A string of values, separated by commas, to be passed to the link editor. These are in addition to those implicitly provided as described in the **cc** command.

ldopt A string naming optional flags that, when encountered on the **cc** command line, to be passed to the link editor. See **cppopt**.

libraries Flags, separated by commas, to be passed as the last parameters to the link editor as the the default is libraries, the default is **−lrts, −lc**.

osuffix The suffix for object files, the default is **o**.

ssuffix The suffix for assembler programs, the default is **s**.

use Values for attributes are taken from the named stanza in addition to the local stanza. For single-valued attributes, values in the **use** stanza apply if no value is provided in the local stanza (or default stanza). For comma-separated lists, the values from the **use** stanza are added to the values from the local stanza.

Example

```
* CC configuration file:

 default:
      cpp        = /lib/cpp

 * standard cc
 cc:
      use        = DEFLT
      crt        = /lib/crt0.o
      mcrt       = /lib/mcrt0.o
      libraries = -lrts,-lc
      ldflags    = -n,-T0x10000000,-K

 * direct floating point accelerator cc
 fcc:
      use        = DEFLT
      crt        = /lib/crt0.o
      mcrt       = /lib/mcrt0.o
      libraries = -lrts,-lfm,-lfc
      ccomflags = -f
      ldflags    = -n,-T0x10000000,-K

 * standard standalone cc
 scc:
      use        = DEFLT
      crt        = /lib/crt2.o
      cppflags   = -DSTANDALONE
      libraries = -l2
      ldflags    = -H4,-Y4

 * atoc
 vcc:
      ccom       = /lib/ccom
      ccomopt    = -O
      copt       = /lib/copt
      as         = /bin/as
```

```
     ld        = /bin/ld
     cppflags  = -Daiws,-DAIX
     ldflags   = -r,-X,-R4,-H4,-Y4,-T0x60
     crt       = /usr/lib/vrmcrt.o

* common definitions
DEFLT:
     ccom      = /lib/ccom
     ccomopt   = 0f
     copt      = /lib/copt
     as        = /bin/as
     ld        = /bin/ld
     cppflags  = -Daiws,-DAIX
     ldflags   = -e,start,-X
```

File

/etc/cc.cfg

Related Information

In this book: "getopt" on page 3-214 and "attributes" on page 4-20.

The **as, cc, ccp**, and **ld** commands in *AIX Operating System Commands Reference.*

connect.con

Purpose

Controls communication connections and data transfer.

Description

The connection configuration file, **/usr/lib/INnet/connect.con** or **$HOME/bin/connect.con**, controls the setup of connections for the **connect** program and for certain optional communications programs. It provides a very general, flexible mechanism to specify how connections are made and how data is transferred after making a connection.

The **connect.con** files are attribute files. The following attributes may appear in the connection control file.

Connection Options

The connection options and their descriptions are:

prefix, address, suffix

The telephone number to dial or the network address to contact. The actual number is constructed by concatenating the prefix (if any), the address, and the suffix (if any). Usually the **prefix** and **suffix** are defined in **/etc/ports** because they depend on the peculiarities of the dialer, and the **address** is defined in **connect.con**.

Multiple addresses can be specified by consecutive **address** assignment lines or by multiple **address** values separated by commas. The addresses are tried in the order given. To specify a comma as part of the command that is sent to the modem, enclose the entire **address** value in quotation marks.

connect Type of connection to make. This option is specified in **/etc/ports** since it is usually associated with the hardware configuration of the outgoing line. Permissible values are:

permanent The connection is hard-wired. No dialing or other special attention is needed.

manual The connection must be completed manually. This generally implies a modem that does not dial, for example, an acoustic coupler.

hayes_1200 The line has a Hayes Stack Smartmodem 1200.

hayes_2400 The line has a Hayes Stack Smartmodem 2400.

vadic The line has a Racal-Vadic 3451P autodialer.

ventel The line has a Ventel MD212+ autodialer.

other_name The line is associated with a dialer program, which is not built into the **connect** program. This option allows you to augment the capabilities of the **connect** program and other communications programs when dealing with new types of communications lines and dialers. The program searches for the named dialer program in **/usr/lib/INnet/dialers** or **$HOME/bin**.

The assumptions made for dialer programs you supply are: the port to be used can be opened prior to dialing and the file will be opened as descriptor 3. Two parameters are passed: number to dial as parameter 1, and dialer hardware to use or value of the dialer option, if any as parameter 2. Any code exit from the dialer except 0 indicates the dialer failed. The failure code returned by the dialer determines the message printed by the programs.

linetype Type of communication line protocols, either **synchronous** or **asynchronous**. Different protocols are used on different line types, so the talker programs may differ. The default linetype is **asynchronous**.

type The name invoked with the **connect** program that determines the kind of connection attempted. Only those stanzas with the proper type are processed. Currently, the **connect** program itself uses only **terminal** type stanzas. The default type is **terminal**.

use This option directs the **connect** program to read the named stanza and follow the instructions there.

Line Options and Parameters

Line options and parameters used are:

min The minimum value to use in kernel buffering. **Min** value characters must be received before a call to the **read** system call returns, unless value specified in **time** elapses.

parity The line is checked for the indicated parity: **even, odd, any,** or **none**.

speed The transmission speed, generally 110, 300, 1200, 2400, 9600, and so on.

time The value to use in kernel buffering. **Time** in tenths of a second to receive a character before a call to the **read** system call returns unless **min** characters are received. See the discussion of ICANON in "termio" on page 6-114. Setting these parameters can result in improved performance.

timeout The time limit to complete the connection in seconds. When the time limit expires, the connection is aborted. This attribute is not needed for devices with a built-in timeout.

System Options

The system options are:

device The name of the special file to use to make the connection. The device must appear in **/etc/ports** (see "ports" on page 4-117) and the information in the ports file entry that is made available to the **connect** program. Note that this attribute can appear only in the last of the list of stanzas associated with making the connection on this device, and that the **use** option must not appear.

dialer This option specifies the dialer hardware to be used in dialing the number. It is normally in **/etc/ports** file, associated with the device to be used. It may also be specified in a connection file, so that its value can be passed to a user-specified dialer program.

Diagnostics

The following diagnostics are displayed, based on the return value from system- or user-supplied dialer programs. The values 8 through 14 are treated as fatal errors.

Code	Message
0	Connected
1	Cannot open dialer
2	Busy or no answer
3	Not able to fork
4	Terminated attempts
5	Communication failure
6	Busy
7	No answer
8	Dead phone
9	Bad phone number
10	Cannot open device specified
11	Address not specified
12	Bad **connect.con** format
13	Cannot run dialer
14	No autodialer specified.

Login Script

A login script is file with the given name that is interpreted prior to notifying you that the connection is complete. Script files are located either in the **$HOME/bin** file or in the **/usr/lib/INnet/scripts** file.

script A script file is organized into a group of states. In each state, the script file optionally sends a string to the remote system, then waits for a response. Several possible responses can be listed for each state along with an action to be performed if the response is received. A time limit can also be set in each state, along with an action to be performed if the time expires without an expected string arriving. The actions are to terminate script interpretations, with either a success or failure indication, or to move to another state. A sample script is shown under "Example" on page 4-37.

DONE
A successful termination of script interpretation.

ERROR *string*
An unsuccessful termination of script interpretation. The last message received from the remote site is reported to you.

GOTO *n*
Continues processing in state *n*.

RECV *string action*
This action is performed if the given string is received.

SEND *string*
Sends the given string to the remote system. Any name enclosed in braces in the string is taken to be an option reference and is replaced by the value of that option. If that option is not present in the list of stanzas, you are prompted for its value using the option name as the prompt. If a - (dash) precedes the name within the braces, the typed characters are not echoed. This is handy for including passwords as parameters in the script file without having them stored on the system. Thus, script files can be given parameters so that they can be used in different connection stanzas and by different users.

STATE *n*
Declares the beginning of state *n*.

TIMER *n action*
This action is performed if no expected string is received in *n* seconds.

Talker Program

A talker program handles the work of moving data across a connection. This program runs after a connection is established. The default talker for the **connect** program is **atalk**. You can override this and specify your own talker program.

talker This is name of the program to run when the connection is made. The conventions observed between the **connect** program and the talker are not complex: the connection is opened by the program as file descriptor 3. The only flag passed by **connect** to the talker program is:

-l*lockfile*

Note: If the **-l** flag is present, the talker must remove the named *lockfile* to make the port available to other users.

flags This option passes flags (other than the above) to the talker program. This option is valid with both default or user-specified talkers.

Example

A typical script might be:

```
STATE 0   RECV User:               GOTO 1
          TIMER 10                 ERROR "No login"

STATE 1   SEND "{myname}\n"
          RECV Password:           GOTO 2
          RECV "Unknown:"          ERROR "Name unknown"
          TIMER 10                 ERROR "No password msg"

STATE 2   SEND "{-mypass}\n"
          RECV "$"                 DONE
          RECV Invalid             ERROR "Wrong password"
          TIMER 20                 ERROR "No prompt"
```

This script could be used for login to a remote RT PC system. In this file, **connect** waits up to 10 seconds for a `User:` prompt. When received, it sends the value of the *myname* option from the control file or by prompt, as the user name. It then waits for 10 seconds for the `Password:` prompt, then it sends the value of *mypass* as the password. The password is not echoed. It then waits another 20 seconds for another prompt. At each stage, it looks for messages that could occur if the given user name or password is invalid. With more states, you can write a script that tries several different user names and types the necessary information to dial through a port selector.

Files

/usr/lib/INnet/connect.con
$HOME/bin/connect.con

Related Information

In this book: "attributes" on page 4-20, "ports" on page 4-117, and "termio" on page 6-114.

The **connect** and **uucp** commands in *AIX Operating System Commands Reference.*

INmail/INnet/FTP.

core

Purpose

Contains an image of memory at the time of an error.

Synopsis

#include <core.h>
#include <sys/param.h>
#include <sys/reg.h>
#include <sys/user.h>

Description

The system writes a memory image of a terminated process when various errors occur in a **core** file in the current directory. See the **signal** system call for the list of errors. The most common are memory address violations, illegal instructions, bus errors, and user-generated quit signals. The memory image, called **core**, is written in the process working directory. A process with an effective user ID that is different from the real user ID does not produce a memory image.

The first section of the memory image is a copy of the system data per user process, including the contents of the registers as they exist at the time of the fault. The size of this section depends on the **usize** parameter defined in **/usr/include/sys/param.h**. The first section contains two parts. The first part is the user structure defined in **/usr/include/sys/user.h**. The second part is the process kernel stack. Note that RT PC stores the user process registers at the beginning of the user stack, instead of the end of the process kernel stack where they are normally stored on machines with stack push and pop instructions. The **/usr/include/sys/reg.h** structure outlines the long word offsets of the registers from the beginning of the user structure. The second section represents the actual contents of the user area when the image was written. If the text segment is separated from data space, it is not dumped.

File

core

Related Information

In this book: "setuid, setgid" on page 2-129 and "signal" on page 2-145.

The **crash** and **sdb** commands in *AIX Operating System Commands Reference.*

cpio

Purpose

Describes copy in and out (cpio) archive file.

Description

When the **-c** flag of the **cpio** command is not used, the header structure is:

```
struct {
  short
     h_magic,
     h_dev;
  unsigned short
     h_ino,
     h_mode,
     h_uid,
     h_gid;
  short
     h_nlink,
     h_rdev,
     h_mtime[2],
     h_namesize,
     h_filesize[2];
  char
     h_name[n];         /* described below */
} Hdr;
```

When the **cpio** command is used with the **-c** flag, the header for the **cpio** structure may be read as:

```
sscanf(Chdr,"%6ho%6ho%6ho%6ho%6ho%6ho%6ho%6ho%11lo%6ho%11lo%s",
&Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
&Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
&Longtime, &Hdr.h_namesize, &Longfile, &Hdr.h_name);
```

Longtime and **Longfile** are equivalent to **Hdr.h_mtime** and **Hdr.h_filesize**, respectively. The contents of each file together with other items describing the file are recorded in an element of the array of varying length structures. The member **h–magic** contains the constant octal 070707 (or 0x71c7). The **stat** system call explains the meaning of structure members **h–dev** through **h–mtime**. The length of the null-terminated path name, *h–name*, including the null byte is indicated by **n**, where **n** = (*h–namesize % 2*) + *h–namesize.* In other words, **n** is equal to *h–namesize* if *h–namesize* is even. If *h–namesize* is odd, **n** is equal to *h–namesize* + 1.

The last record of the archive always contains the name `TRAILER!!!` Special files, directories, and the trailer are recorded with **h–filesize** equal to 0.

Related Information

In this book: "stat, fstat" on page 2-159 and "scanf, fscanf, sscanf, NLscanf, NLfscanf, NLsscanf" on page 3-325.

The **cpio** and **find** commands in *AIX Operating System Commands Reference.*

ddi

Purpose

Contains device-dependent information (ddi).

Description

A **ddi** file contains information for customizing classes (or types) of hardware adapters or devices supported by the system. The information in this file may be modified using the **devices** command or an editor program. The **ddi** files are attribute files that are located in the **/etc/ddi** directory. See "attributes" on page 4-20 for the format of attribute files.

The equivalent of a **ddi** file can be created and added to the system. ***Customize helper*** programs convert the parameters in the files into a corresponding **Define_Device** structure, which is used by the VRM device driver. A **ddi** file contains the following information:

- Device-dependent information. This is a series of keywords whose values the user supplies when the device is defined.
- Instructions to the customize helper program for processing input parameters
- Mapping information for the **ddi** structure.
- Bit field mapping information.

The use of extended characters in **ddi** files is not supported.

Keywords

The following keywords are used in the stanzas of device-dependent information files. These keywords describe attributes and settings for a particular device that may be changed to suit your device.

Key Word	**Description**	**Possible Choices**
aa	**Automatic Answering**: Does the printer support communication auto answering?	true, false
ae	**Automatic Enable**: Do you want the port to be enabled automatically?	true, false

Key Word	Description	Possible Choices
alf	**Automatic Line Feed**: Does the printer have automatic line feed with carriage return.	true, false
ami	**Adapter Microcode IOCN**: The IOCN of the microcode module for an adapter that requires an IPL.	
appt	**Application Type**: The type of application that runs on the link.	3280, rje
ars	**Aspect Ratio Support**: Does the printer have a set aspect ratio control?	yes, no
at	**Adapter Type**: Refers to the last two digits of the service request number.	
backs	**Backspace Support**: Does the printer have the ability to backspace (move print head backward while printing a line)?	yes, no
bigs	**Bit-Image Graphics Support**: Does the printer have bit image graphics controls?	yes, no
biopa	**First I/O Port Address**: Refers to the hardware adapter address.	
bm	**Bottom Margin (last line)**: Refers to the number of the last writing line.	1 - [length(in.) × lines/in.]
bpc	**Bits Per Character**: Refers to the character length in bits.	5, 6, 7, 8
brea	**Bus RAM End Address**: If not zero, refers to the adapter's end RAM address on the I/O bus.	
brsa	**Bus RAM Start Address**: If not zero, refers to the adapter's start RAM address on the I/O bus.	
bs	**Block Size**: Refers to the size of sectors of storage on the fixed disk; changes in number of blocks on minidisks may affect system performance.	512, 1024, 2048

Key Word	Description	Possible Choices
bsess	**Base Session**: Indicates the base session number for the link.	
cdp	**Condensed Print**: Does the printer support printing in condensed characters?	yes, no
cdpg	**Code Page**: Specifies the code page loaded into the printer.	437 (PC), 850 (MLP)
cn	**DMA Channel Number**: Refers to DMA channel number.	0 - 9
colp	**Color Printer**: Refers to whether the printer is capable of printing in color.	yes, no
cps	**Condensed Print Support**: Can the printer print in condensed characters?	yes, no
cp1 - cp8	**Code Page 1 through Code Page 8**: Specifies the code pages loaded into the printer. The IBM 5201 Printer and 4201 Proprinter use only **cp1** and **cp2**.	PC, A, B, C, D, P0, P1, P2, MLP
cr	**Color Ribbon**: Does the printer have color ribbon?	yes, no
cs	**Character Set**: Refers to complete groups of characters that a printer can support.	1, 2
cus	**Continuous Underscore Support**: Supports the escape - (minus) control.	yes, no
ddbw	**Device Data Bus Width**: Is the I/O adapter an 8-bit or 16-bit device.	8, 16
dmao	**Uses DMA Support Only**: Use only DMA transfers.	true, false
dmas	**DMA Support**: Refers to whether hardware adapter supports DMA.	true, false
dnec	**Do Not Enable DMA Channel**: The channel is not enabled if the value is **true**.	true, false

Key Word	Description	Possible Choices
dpc	**Default Print Color**: Refers to the color to use for printing when a file does not contain codes that specify a color: usually black, blue, red, or yellow.	black, blue, red, yellow
dsp	**Double Strike Print**: Should double-strike be turned on?	yes, no
dsps	**Double Strike Print Support**: Does the printer have a control double-strike characters and provide boldface?	yes, no
dvam	**Device Attachment Method**: Is your device attached locally with a cable or is it attached through a modem?	0 = local, 1 = remote (modem)
dwp	**Double Width Print**: Should a file be printed with a double-width character set?	yes, no
dwps	**Double Width Print Support**: Does the printer have the ability to print with a double-width character set?	yes, no
ei	**Enable Adapter Interrupts.**	
ei1 - ei4	**Enable First through Fourth Interrupt Level**: Refers to conditions allowing the printer to stop when an error occurs or when assistance is needed to complete I/O on interrupts levels 1, 2, 3, or 4.	true, false
ep	**Emphasized Print**: Should emphasized print be turned on? Every character is overstruck with a second pass of the print head.	yes, no
eps	**Emphasized Print Support**: Does the printer have a control to do emphasized print?	yes, no
fd	**Fixed Disk**: Refers to one of up to three circular plates used for storing data.	hdisk0, hdisk1, hdisk2
fi	**Frequency Input**: Refers to clock frequency to USART chips.	
fid	**Font ID**: ID of the font used by the printer.	11

Key Word	Description	Possible Choices
fl	**Form (page) Length**: Refers to the length of the paper in terms of the number of lines per page. The value is determined by multiplying the length of paper (in inches) by the number of lines printed per inch.	1 - [(in.) × lines/in.]
fnt1 - fnt8	**Font 1 through Font 8**: Specifies the printer fonts, such as letter gothic or prestige elite. This keyword is used for the IBM 3812 Pageprinter.	
fp	**First Party DMA**: Refers to whether hardware adapter has own DMA controller.	true, false
fw	**Form Width (right margin)**: Refers to the width of paper in terms of the number of characters per line. The value is determined by multiplying the width of the paper (in inches) by the number of characters printed per inch.	1 [width(in.) × pitch]
hsi	**Horizontal Spacing Increment**: What horizontal increment is used in the ESC K control?	60, 70, 120
hts	**Horizontal Tab Support**: Does the printer have horizontal tab controls?	yes, no
htvi	**Text Vertical Increment**: The vertical index increment used by subsequent CUU (ESC A) multi-byte controls. (See "Multi-Byte Controls" on page 5-13.)	72
ic1 - ic4	**Service class of First through Fourth Interrupt**: Refers to interrupt priority, where 0 is the highest.	0, 1, 2, or 3
il1 - il4	**Interrupt Level Number of First through Fourth Interrupt**: Refers to hardware adapter interrupt levels 1 through 4.	
iobd	**Input/Output Bus Device.**	true, false
ioccb	**Using DMA Four-Byte Buffering.**	true, false
iof1 - iof8	**Read/Write Flag for I/O Operation 1 through 8** : Is a read or write required to the corresponding I/O port address (pa1 - pa8)?	0 = Input, 1 = Output

Key Word	Description	Possible Choices
iopar	**Number of I/O Port Addresses**: Refers to hardware adapter address range.	
iow1 - iow8	**I/O Width for I/O Operation 1 through 8**: Refers to the number of bits to be written to or read from the port address (pa1-pa8).	0 = 8 bit, 1 = 16 bit
ip	**Initialize Printer**: Refers to the initial state of the printer after power is applied.	true, false
ixp	**Include Xon/Xoff Protocol**: Refers to whether communication protocol is included.	true, false
js	**Justification Support**: Refers to printing with the right margin even.	yes, no
kpoe	**Keep Printing on Error**: Should the printer complete the print job despite errors (without sending an error message to the user)?	yes, no
llo	**Leave DTR and RTS Lines On.**	true, false
lm	**Left Margin**: Refers to the area on a page between the left edge and the leftmost character position on the page.	0 - [width(in.) × pitch]
lobibp	**Length of Buffers in Buffer Pool**: The length in bytes of each buffer in the buffer pool of the Block I/O Communication Area (BIOCA).	
logger	**PTY supports a login shell.**	true, false
lpi	**Lines Per Inch**: Refers to the number of print lines per inch, to line spacing density, and to the distance paper moves during a line feed.	6, 8
lrmc	**Left/Right Margin Controls**: Does the printer have the ability to change left and right margins (does it have left and right margin control codes)?	yes, no
lun	**Logical Unit Number**: Number associated with an addressable physical or logical device.	0 - 7

Key Word	Description	Possible Choices
mask	**Mask**: Refers to the mask that indicates defaults overridden by corresponding field in the **Define–Device** structure.	F1FC
mc	**Modem Controls**: Refers to whether or not to enable modem controls.	true, false
mccs	**Multibyte Control Code Support**: If **yes**, then the printer supports IBMOEM multi-byte controls. If **no**, then the printer is assumed to function like an IBM 5152 printer.	yes, no
mnoal	**Maximum Number of Attached LCCs**: The maximum number of LCCs that can be attached to the device driver using the Block I/O Communication Area (BIOCA).	
mnonid	**Maximum Number of Net IDs**: The maximum number of network IDs that the device driver can support.	
nidd	**Net ID Displacement**: The offset (in bytes) into the receive data of the network ID.	
nidl	**Net ID Length**: The length in bytes of a network ID.	
noabb	**Number of Allowed Bad Blocks**.	
nob	**Number of Blocks**: Refers to the number of blocks in a minidisk.	
nobibp	**Number of Buffers in Buffer Pool**: The number of buffers to be allocated in the buffer pool of the Block I/O Communication Area (BIOCA).	
nobod	**Number of Blocks on Device**.	
nobodr	**Number of Buffers on a Device Ring**: The number of buffers to be allocated for each device ring queue in the Block I/O Communication Area (BIOCA).	
nobub	**Number of 256-Byte Units/Block**: Number of 256-byte units on each block.	

Key Word	Description	Possible Choices
noi	**Number of Interrupt Levels Used**: Refers to the number of hardware interrupt levels.	
nops	**Number of I/O Operations.**	1 - 8
nor1 - nor8	**Number of Repetitions for I/O Operation 1 through 8**: Refers to the number of times the same I/O operation is performed to the corresponding port address (pa1 - pa8).	1 - 64
norbosr	**Number of Receive Buffers on SLIH Ring** queue.	
nosb	**Number of Stop Bits**: Refers to the number of stop bits in a communication character.	1, 1.5, 2
nospt	**Number of Sectors per Track.**	
now	**Number of DMA Sub-Channels.**	0
nr	**No Read-Only Memory.**	true, false
nsess	**Maximum Number of Sessions**: The maximum number of sessions that can be run on the link.	1, 2, 3, 4, 5, 6, 7, 8
od1 - od8	**Output Data for I/O Operation 1 through 8**: Refers to the data to be written to the corresponding port address (pa1 - pa8) if the corresponding flag (iof1 - iof8) is set for output.	0000 - FFFF
om	**Operation Mode**: Refers to whether communications operation mode is set.	tx, rx, full, half
pa1 - pa8	**Port Address for I/O Operation 1 through 8**: Refers to the adapter port address being written to or read from to disable the adapter.	0000 - FFFF
pacs	**Print All Characters Support**: Does the printer support ESC ^ and ESC \ controls?	yes, no
pdt	**Peripheral Device Type.**	

Key Word	Description	Possible Choices
ph	**Paper Handling**: Refers to the way the printer handles different types of paper. The manual-feed printer stops at the end of each page and waits for the user to insert another sheet and press the start button. A printer with an automatic sheet-feed mechanism feeds paper to the printer.	0 = manual; 1 = automatic; 2 = continous form paper.
pinit	**User-Supplied Sequence**: Refers to the control sequence the co-processor to reset the printers whose fonts can be changed.	
pitch1 - pitch8	**Character Pitch 1 through Character Pitch 8**: Refers to the number of characters per linear inch, for instance, 10-pitch type has 10 characters per inch.	10, 12, 15
plot	**Pass Data Directly to Device Without Modification.**	yes, no
pn	**Port Number on Adapter**: Refers to the hardware adapter port.	0 - 4
pq	**Print Quality**: May select (on some printers) degrees of print quality: **dp** (for fast, low quality), **text** (for better draft quality), **letter** (for high-quality final text).	dp, text, letter
prin	**Printer Type**: 0 = unspecified (functionally 5152); 1 = IBM 5152; 2 = IBM 5182; 3 = reserved; 4 = IBM 5201 Printer; 5 = IBM 4201 Proprinter; 6 = IBM 4202; 7 = IBM 3852.	0, 1, 2, 3, 4, 5, 6, 7
pro	**Protocol**: Refers to communication protocol.	dtr, cdstl, dc
psd	**Paper Source Drawer**: Refers to the location of the paper drawer from which paper is drawn for printing.	1 = top; 2 = bottom
pss	**Proportional Spacing Support**: Does the printer support proportionally spaced printing?	yes, no
pt	**Parity Type**: Refers to communication character parity.	even, odd, mark, space, none
rdto	**Receive Data Transfer Offset**: The device driver using block I/O transfers the receive packet beginning at this offset into the buffer.	

Key Word	Description	Possible Choices
rea	**Bus ROM End Address**: If not zero, refers to the adapter's start ROM address on the I/O bus.	
rl	**RAS Length**: The length in words of the RAS section of the Define Device structure.	
rlfs	**Reverse Line Feed Support**: Does the printer support the ESC] control?	yes, no
roffv	**Receive Xoff Value**: Refers to character to transmit in order to inform a remote device to stop sending data.	00 - FF
ronv	**Receive Xon Value**: Refers to character to transmit in order to inform a remote device to resume sending data.	00 - FF
rsa	**Bus ROM Start Address**: If not zero, refers to the adapter's end ROM address on the I/O bus.	
rtrig	**Receive Buffer Trigger**: If the adapter has receive data buffering capability, then this value selects the number of bytes that trigger a received data interrupt.	1, 4, 8, 14
rts	**Receive/Transmit Speed**: Refers to communication baud rate.	50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200
rxt	**Receive Xoff Threshold**: Refers to threshold for full communication buffer detection.	20
sa	**Strobe Active**.	true, false
sdmac	**Shared DMA Channel**: Refers to whether a hardware adapter can share DMA channel.	true, false

Key Word	Description	Possible Choices
sg	**DMA Scatter/Gather Support**: Refers to DMA support the ability of hardware to scatter and gather I/O data.	true, false
si1 - si4	**Share First through Fourth Interrupt Levels**: Refers to whether interrupt levels 1, 2, 3, or 4 are able to be shared.	true, false
sid	**SCSI ID**: Refers to the SCSI ID number.	0 - 6
slap	**Skip Lines at Perforation**: Refers to the number of lines skipped at page breaks. The number is divided by 2, so that half the blank lines appear at the bottom of one page and half at the top of the next.	0-[length(in.) × lines/in.]
slow	**Slow Device Support**: Refers to whether DFT slow device support is enabled.	0 = Disabled, 1 = Enabled
sn	**Slot Number**: Refers to the slot in which an adapter is installed.	1 - 8
sns	**Switched/Nonswitched**: Refers to the state of the communication line connection.	true, false
sp	**Select Printer.**	true, false
sppt	**Serial/Parallel Printer Type**: Refers to whether the printer is a serial or parallel type.	1 = Parallel, 2 = Serial
srbt	**SLIH Ring Buffer Threshold**: The number of SLIH ring queue buffers that the device driver can use before requesting additional buffers from the block I/O device manager.	
sss	**Superscript/Subscript Support**: Does the printer have the ability to print in superscript and subscript mode?	yes, no
sysadd	Specifies the action that the **devices** command takes after adding the device. The valid choices are: **a** Rebuilds the kernel and IPLs the system **v** Runs the **vrmconfig** command **none** Takes no special action.	a, v, none

Key Word	Description	Possible Choices
sysdel	Specifies the action that the **devices** takes after deleting the device. The valid choices are: **a** Rebuilds the kernel and IPLs the system **v** Runs the **vrmconfig** command **none** Takes no special action.	a, v, none
tbc	**Transmit Buffer Count**: Number of bytes to buffer for transmitter.	
tm	**Top Margin**: Refers to the number of lines to be skipped at the top of a page before printing begins. If the user specifies 6 lines, the first print line will be line 7. The value is determined by the length of paper (in inches) multiplied by the number of lines per inch.	0 - [length(in.) × lines/in.]
toffv	**Transmit Xoff Value**: Refers to the communication character to transmit in order to inform a remote device to cease sending data.	00 - FF
tonv	**Transmit Xon Value**: Refers to the communication character to transmit in order to inform a remote device to resume sending data.	0 - FF
tt	**Terminal Type**: Refers to the type of the terminal being used.	
type1 - type8	**Typestyle 1 through Typestyle 8**: Refers to a typestyle such as bold or italic.	
urpim	**User to Receive Printer Intervention Messages**: Refers to whether printer intervention messages are sent to any valid user or to the user who queued the print job.	Any user ID, pjo = Print Job Owner
vhs	**Variable Horizontal Spacing**: Does the printer have ESC d and ESC e controls?	yes, no
vpqs	**Variable Print Quality Support**: Does the printer have the ability to print different degrees of quality?	yes, no
vsi	**Vertical Spacing Increment**: Refers to parts of inch supported in ESC 3 and ESC J controls.	216, 144

Key Word	Description	Possible Choices
vts	**Vertical Tab Support**: Does the printer support vertical tabs?	yes, no
wll	**Wrap Long Lines**: Does the printer "wrap" lines? That is, will it break lines longer than the specified form width at the right margin and print the remainder on the next line?	yes, no
12ps	**12 Pitch Support**: Does the printer support 12 pitch?	yes, no

Files

/etc/ddi/diskette
/etc/ddi/enet
/etc/ddi/float
/etc/ddi/font
/etc/ddi/opprinter
/etc/ddi/plotter
/etc/ddi/pprinter
/etc/ddi/sprinter
/etc/ddi/tty

and possibly others.

Related Information

In this book: "attributes" on page 4-20, "master" on page 4-98, "system" on page 4-139, "descriptions" on page 4-56, "kaf" on page 4-94, "options" on page 4-110, and "predefined" on page 4-124.

descriptions

Purpose

Describes the meaning of **ddi** file keywords.

Description

The **/etc/ddi/descriptions** file contains a sorted list of descriptions for each of the keywords used in **ddi** files. The **devices** command uses this file to explain the meanings of the keywords during the **add**, **change**, and **showdev** subcommands.

The **/etc/ddi/descriptions** file must be sorted by keyword, and each line must follow the following format:

keyworddescription

where:

keyword Names a keyword that is used in a **ddi** file. This field is exactly 10 characters long, is padded on the right with spaces, and contains no tabs.

description Describes the meaning of the keyword. This field is exactly 28 characters long, is padded on the right with spaces, and contains no tab characters.

Note: The **/etc/ddi/descriptions** file must be sorted alphabetically by the *keyword* field. If it is not sorted, then the **devices** commands displays incorrect information about the meanings of keywords.

The use of extended characters in the **/etc/ddi/descriptions** file is not supported.

File

/etc/ddi/descriptions

Related Information

In this book: "ddi" on page 4-43 and "options" on page 4-110.

devinfo

Purpose

Contains device characteristics.

Synopsis

#include <sys/devinfo.h>

Description

The **devinfo** structure is defined for each device. The **IOCINFO** operation of the **ioctl** system call fills in this structure. The information returned by a device varies. Most devices, other than disk devices, return a **devtype** value and the remainder of this structure contains zeros. This structure provides information about the capabilities of a device, rather than its current status or settings. For example, types of information provided are the number of characters a printer handles per line or the diskette capacity in number of blocks.

The maximum size of this structure is 12 bytes (no longer than the disk version), so that programs can use the **ioctl** system call without concern of overrun due to increasing size.

```
struct devinfo
{  char devtype;
   char flags;
   union
   {
      struct                /* for disks */
      {  short  bytpsec;    /* bytes per sector */
         short  secptrk;    /* sectors per track */
         short  trkpcyl;    /* tracks per cylinder */
         long   numblks;    /* blocks this minidisk */
      } dk;
      struct                /* for memory mapped displays */
      {
         char  capab;       /* capabilities */
         char  mode;        /* current mode */
```

```
                short  hres;      /* horizontal resolution */
                short  vres;      /* vertical resolution */
            } tt;
        } un;
    };
```

The following flags specify some generic capabilities (see DD_DISK):

Constant	Value	Function
DF_FIXED	01	Not removable
DF_RAND	02	Random access possible
DF_FAST	04	A relative term

The **devinfo** structures are defined for the following devices (specified in the **devtype** field):

DD_DISK Indicates a disk. This **devtype** is **R**. The driver determines the values. The fixed disk has flags DF_RAND | DF_FIXED | DF_FAST, while the diskette has flags DF_RAND (see "fd" on page 6-17 and "hd" on page 6-20).

The number of the bytes per sector, sectors per track, and tracks per cylinder for the fixed disk are predetermined. The minidisk table determines the number of blocks. For the diskette, the minor device driver or the physical media determines this information when the device is opened.

DD_LP Indicates a line printer. The **devtype** is **l**. This fills in the **devtype** field and returns zeros for the rest of the structure.

DD_PSEU Indicates a pseudo-device. This **devtype** is **Z**.

DD_RTC Indicates a real-time (calendar) clock. This **devtype** is **c**.

DD_TAPE Indicates a magnetic tape. This **devtype** is **M**.

DD_TTY Indicates a terminal. This returns a **devtype** of **T** and zeros for the rest of the structure.

DT_STREAM Indicates a streaming tape drive. The **devtype** is **2**.

DT_STRTSTP Indicates a start-stop tape drive. The **devtype** is **2**.

Related Information

In this book: "ioctl" on page 2-56, "fd" on page 6-17, and "hd" on page 6-20.

dir

Purpose

Describes the format of a directory.

Synopsis

#include <sys/dir.h>

Description

A directory is a file that a user is not allowed to write into directly. A directory file contains a 16-byte entry for each file in it. A bit in the flag word of the i-node entry indicates that the corresponding file should be treated as a directory. For additional information about a system volume format, see the "fs" on page 4-74. The structure of a directory entry as given in the include file is:

```
#include <sys/types.h>
#ifndef  DIRSIZ
#define  DIRSIZ  14
#endif
struct  direct
{
   ino_t   d_ino;
   char    d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are . (dot) and .. (dot dot). The first . (dot) is an entry for the directory itself. The .. (dot dot) entry is for the parent directory. The meaning of the **..** (dot dot) entry for the root directory of the master file system is modified. There is no parent, directory, therefore, the **..** (dot dot) entry has the same meaning as **.** (dot).

Related Information

In this book: "fs" on page 4-74 and "inode" on page 4-92.

errfile

Purpose

Contains system event log.

Synopsis

#include <sys/erec.h>

Description

When a system event occurs and logging is active, it generates an event record and passes the record to the event-logging daemon to be recorded in the event log. The **/etc/rasconf** file specifies the files where the events are to be logged. The default event log file is **/usr/adm/ras/errfile**.

Every record has a header. See "error" on page 6-15 for the structure of a header. Each type of event record has its own format. The **/usr/include/sys/erec.h** file shows the format of the events currently logged. The error daemon process gathers the records from memory and writes them in the files on disk. The event log file is opened (if existing) or created. Next, the process opens the **/dev/error** special file, formats and writes the non-volatile random access memory (NVRAM), which can contain up to 16 bytes of information, and reads the events logged in memory. An analysis routine is called before an event is written to the **errfile**. For an error, this routine returns a buffer of probable cause information to aid in problem determination. This buffer is appended to the error entry, the length of the entry is adjusted, and then the entire entry is written to the file.

Some records in the event file are administrative. These include the startup record entered when logging is activated, the stop record written if the daemon is terminated gracefully, and the time-change record that accounts for changes in the system time of day.

Files

/usr/adm/errfile
/dev/error
/etc/rasconf

Related Information

In this book: "error" on page 6-15 and "rasconf" on page 4-133.

The **errdemon** in *AIX Operating System Commands Reference.*

filesystems

Purpose

Centralizes file system characteristics.

Description

A file system is a complete directory structure, including a root directory and any directories and files beneath it. A file system is confined to a single partition. All of the information about the file system is centralized in the **filesystems** file. Most of the file system maintenance commands take their defaults from this file. The file is organized into stanzas whose names are file system names and whose contents are attribute-value pairs specifying characteristics of the file system.

The **filesystems** file serves two purposes:

- It documents the layout characteristics of the file systems.
- It frees the person who sets up the file system from having to enter and remember items such as the device where the file system resides because this information is defined in the file.

File System Attributes

Each stanza names the directory where the file system is normally mounted. The attributes specify all of the parameters of the file system. See "attributes" on page 4-20 for the format of an attribute file. The attributes currently used are:

account Used by the **dodisk** command to determine the file systems to be processed by the accounting system. This value can be either **true** or **false**.

backupdev Used by the **backup** and **restore** commands to determine the default output device associated with each file system. The value of this keyword is usually the name of a diskette or magnetic tape special file.

backuplen Used by the **backup** command to determine the size of the default backup device associated with each file system. The size of a tape is measured in tracks times feet. For example, the **backuplen** for a 300-foot 9-track tape is 2700. This parameter is ignored for diskettes.

backuplev Used by the **backup** command to determine the default backup level to take for each file system. Backup levels are discussed in the **backup** command.

boot Used by the **mkfs** command to initialize the boot block of a new file system. This specifies the name of the load module to be placed into the first block of the file system.

check Used by the **fsck** command to determine the default file systems to be checked. **true** enables checking while **false** disables checking. If a number, rather than **true** is specified, the file system is checked in the specified pass of checking. Multiple pass checking, described in **fsck** command in *AIX Operating System Commands Reference*, permits multiple file systems to be checked in parallel when multiple drives exist.

cluster Specifies the number of 512-byte disk blocks that the system treats as a unit. Only one or two values are supported. The RT PC default values are **4** for non-removable disks and **1** for removable disks.

cyl Used by the **mkfs** command to initialize the free list and superblock of a new file system. The value is the number of blocks in one cylinder. It defines the size of an interleave cluster.

dev Identifies, for local mounts, either the block special file where the file system resides or the file or directory to be mounted. System management utilities use this attribute to map file system names to the corresponding device names. For remote mounts, identifies the file or directory to be mounted.

free Used by the **df** command to determine which file systems are to have their free space displayed by default. This value is either **true** or **false**.

mount Used by the **mount** command to determine whether or not this file system should be mounted by default. If **mount = true**, then the **mount all** command mounts this file system. If **mount = false**, the file system is not mounted by default. When the optional second value **readonly** is specified, the file system is normally mounted read-only.

Another optional value is **inherit**. When a remote file system is mounted with **mount = inherit**, any additional file systems contained in the specified file system are also mounted. This allows the local node to duplicate the file system structure of the server node, starting at the specified mount point.

In the sample file, notice the line for the root file system that reads **mount = automatic**. The operating system automatically mounts this file system when it is rebooted. The **true** value is not used so that mount **all** will not try to mount it. Also, it is not **false**, because certain utilities, such as **ncheck** normally avoid file systems with **mount = false**.

If **mount = true,removable**, a diskette file system is automatically mounted when its files are opened and unmounted when the opened files are closed. Also notice that in the example, this file is shipped designating two removable file systems, one having asterisks. The asterisks indicate commented lines in the file. The **mkdir** command must be used to create a directory in order to mount file system **/dev/fd1**.

nodename Used by the **mount** command to determine which node contains the remote file system. If this attribute is not present, the mount is a local mount. The value of **nodename** can be either a valid node nickname or a valid node ID.

size Used by the **mkfs** command for reference and to build the file system. The value is the number of blocks in the file system.

skip Used by the **mkfs** command to initialize the free list and superblock of a new file system. The value is the number of blocks to skip when the free list is interleaved. This number is processor- and device-specific.

type Used by the **mount** command to determine whether or not this file system should be mounted. When the command `mount -t` *string* is issued, all of the currently unmounted file systems with a **type** equal to *string* are mounted.

vcheck Used by the **varyon** command to determine which file systems to check. **true** enables checking while **false** disables checking. This keyword should only be set to **true** for filesystems that reside on IBM 9332 Direct Access Storage Devices.

vol Used by the **mkfs** command when initializing the label on a new file system. The value is a volume or pack label using a maximum of six characters. The file system label is always the stanza name.

Example

```
*
* File system information
*

default:
    vol        = "RT PC"
    mount      = false
    check      = false
    free       = false
    backupdev  = /dev/rfd0
    backuplen  = 2400

/:
    dev        = /dev/hd0
    vol        = "root"
```

```
       mount      = automatic
       check      = true
       free       = true

   /u:
       dev        = /dev/hd1
       vol        = "/u"
       mount      = true
       check      = true
       free       = true

   /u/joe/1:
       dev        = /u/joe/1
       mount      = inherit
       nodename   = vance

   /usr:
       dev        = /dev/hd2
       vol        = "/usr"
       mount      = true
       check      = true
       free       = true

   /tmp:
       dev        = /dev/hd2
       vol        = "/tmp"
       mount      = true
       check      = true
       free       = true

   /diskette0:
       dev        = /dev/fd0
       mount      = true,removable

 *  /diskette1:
 *      dev        = /dev/fd1
*      mount      = true,removable
```

File

/etc/filesystems

Related Information

In this book: "attributes" on page 4-20 and "fs" on page 4-74.

The **backup**, **df**, **fsck**, **mkfs**, **mount**, **restore**, and **umount** commands in *AIX Operating System Commands Reference.*

fonts

Purpose

Defines annotated and geometric character fonts for an HFT display device.

Description

The AIX operating system can supply font definitions to the VRM. This can be done either by configuring new font files into the VRM or by dynamically installing font modules into the VRM and issuing an **HFRCONF** operation with the **ioctl** system call to inform the VRM that they exist.

IBM supplies two sets of precompiled annotated text fonts with the AIX Operating System. One set of fonts is for the IBM 5081 Display Adapter and the other set is for all other GSL supported devices. The fonts for the IBM 5081 Display Adapter cannot be used on other devices and fonts for other devices cannot be used on the 5081 Display.

Some of these annotated text fonts are automatically installed with the VRM, and others can be configured into the system by modifying the **/etc/master** file. Also, you can use the **display** command to select the active display font.

GSL supported devices also recognize one geometric text font format that allows you to design your own set of characters. A geometric text font is also known as a programmable character set (PCS) font. The PCS font can be used on all GSL supported devices including the IBM 5081 Display.

In addition to the precompiled fonts, IBM supplies the source for each non5081 font, which you can copy and modify to create new font definitions.

Since the precompiled source files must be linked to the VRM at run-time, these font files must be compiled and converted to table of contents (TOC) format using the **vcc** and **vrmfmt** commands. See *Virtual Resource Manager Technical Reference* for details about the TOC object module format.

An annotated text font definition file has three major parts in the following sequence:

- A header that describes the font. The header is the same for all annotated text fonts.
- A set of character descriptions:
  - 5081 fonts – A set of expanded character bit arrays that describes each character in the font.
  - Non5081 fonts – A set of condensed raster mosaics that describes each character in the font.

- A look-up table that has an index entry to find each character representation in the font.
  - 5081 fonts — look-up table entries are 16 bits.
  - Non5081 fonts — look-up table entries are 32 bits and each describes the start of its raster mosaics entry, its width, and the white space compressed from the top and bottom of its raster mosaics entry.

Annotated Text Font Header

The annotated text font header is a fixed-length structure common to all annotated text fonts for all displays. The VRM run-time binder uses the DDDFSIZE field in the header to link the font to the virtual terminal resource manager. The information in header fields are:

Offset in Bytes	Length in Bytes	Field	Description
0x00	4	DDDFSIZE	The size in bytes of the area containing the font and the look-up table.
0x04	2	fntclass	A number that uniquely identifies the format of the look-up table that follows: 0x01 = not a 5081 font 0x02 = a 5081 font
0x06	2	fntid	The name an application uses to identify a font. This must be a value within the range of 0 to 1024.
0x08	4	fntstyle	Font style.
0x0C	4	fntattr	Identifies the attributes of the font. Possible values are: 0x0000 - no special values 0x0001 - bold version of this font 0x0002 - italic version of this font.
0x10	4	fnttotch	The total number of characters in the font. This is used to determine whether a specified character code is valid for this font.
0x14	4	fnttblsz	Total number of words in the font table.
0x18	2	fntbasln	The scan line within a character box of the baseline for characters in this font (zero origin).

Offset in Bytes	Length in Bytes	Field	Description
0x1A	2	fntcapln	The scan line within a character box of the caps line for characters in this font (zero origin).
0x1C	2	fntcolmn	Width of character box in pels.
0x1E	2	fntrows	Height of character box in pels.
0x20	2	fntchrbt	Total number of bits per character.
0x22	2	fntultop	The scan line within the character box of the top line in the underscore (zero origin).
0x24	2	fntulbot	The scan line within the character box of the bottom line in the underscore (zero origin).
0x26	1	fntmonpt	Mono pitch flag in leftmost bit of this byte.
0x28	4	fntlkup	Byte offset from the beginning of this structure to the beginning of the font look-up table.

Annotated text Font Raster Mosaics (non5081)

This contains a definition for each character in the font. Each character is entered in this area with the horizontal slices bit-packed one right after the other. The first bit of the first character slice is forced to begin in the most significant bit of a byte. The raster mosaics start immediately after the header (0x2C from the start address of the font structure). See *Annotated Text Example One (non5081)* on 4-71.

Annotated Text Character Bit Array (5081)

This contains a definition for each character in the 5081 font. Each character in the character bit array must be a multiple of 4 pels wide and a multiple of 4 pels high. Zeros are padded to the right and padded to the bottom of the character as needed to accomplish this.

Each 4x4 pel array is then stored in a 16-bit word with the first four bits of the array leftmost in the word and proceeding to the right.

The 4x4 arrays are stored beginning with the bottom left array in the character and is repeated across the bottom of the character. The process then continues at the left of the next higher horizontal row of 4x4 arrays and so on until the 4x4 array representing the top right corner of the character is stored in a 16-bit word. See *Annotated Text Example Two (5081)* on 4-72.2.

Annotated Text Font Look-up Table (non5081)

The look-up table immediately follows the raster mosiac. There is one 32-bit look-up table entry for each character in the font. The look-up table can be found by adding the value **fntlkup** given in the header to the starting address of the font structure. The table entry for any given character is found by using the font position number as an index into the table. (See "display symbols" on page 5-24 for a list of the font position numbers.) Each look-up table entry contains the following fields:

Offset in Bits	Length in Bits	Field	Description
0	5	lkup_top	The number of blank scan lines that have been eliminated from the top of this character raster image (white space).
5	5	lkup_bot	The number of blank scan lines that have been eliminated from the bottom of this character raster image (white space).
10	6	lkup_width	Contains the width in pels of this particular character.
16	16	lkup_ref	Byte offset from the start of the the raster mosaics of the first scanline of the character's raster image.

Annotated Text Font Look-up Table (5081)

The character look-up table for IBM 5081 fonts contains an entry for each character or possible character in the font.

Each entry is 16 bits and contains the offset from the start of the character bit array to the first byte of the bit array for the corresponding character. That is,this offset is kept as a 16-bit word offset from the start of the bit array section. The character look-up table entry for any given character is found by concatenating an offset to the start of the code page in the character look-up table with the ASCII (or EBCDIC) character code and adding the result to the starting address of the character look-up table found in the font header.

Annotated Text Example One (non5081)

See Figure 4-1 on page 4-72.1 for this example. The character chosen is a capital **A**. This is shown as it would appear on the display and how it would be stored in the raster mosaics. Also shown is the font look-up table entry for this character. Note that the data associated with the top and bottom two scan lines of the character image do not appear in the raster mosaics since they consist of zeros.

To reconstruct the character image from the raster mosaics, it is necessary to use the font look-up table. The display symbol code associated with the character that is to be displayed is used to access its corresponding 4-byte entry in the font look-up table. The information contained in a font look-up table entry is shown. The capital **T**'s represents the bits containing the number of top blank scan lines that were compressed from the character image. The capital **B**'s represents the bits containing the number of bottom blank scan lines that were compressed from the character image. The capital **W**'s represents the bits containing the width in pels of this character. Capital **O**'s represent the bits containing the offset of the compressed portion of this character image data in the raster mosaics. For this example, the value associated with **T** is 2, the value associated with **B** is 2, and the width (**W**) is 5. The value associated with **O** is the offset of the $\mathbf{y}^{th}$ byte of the raster mosaics.

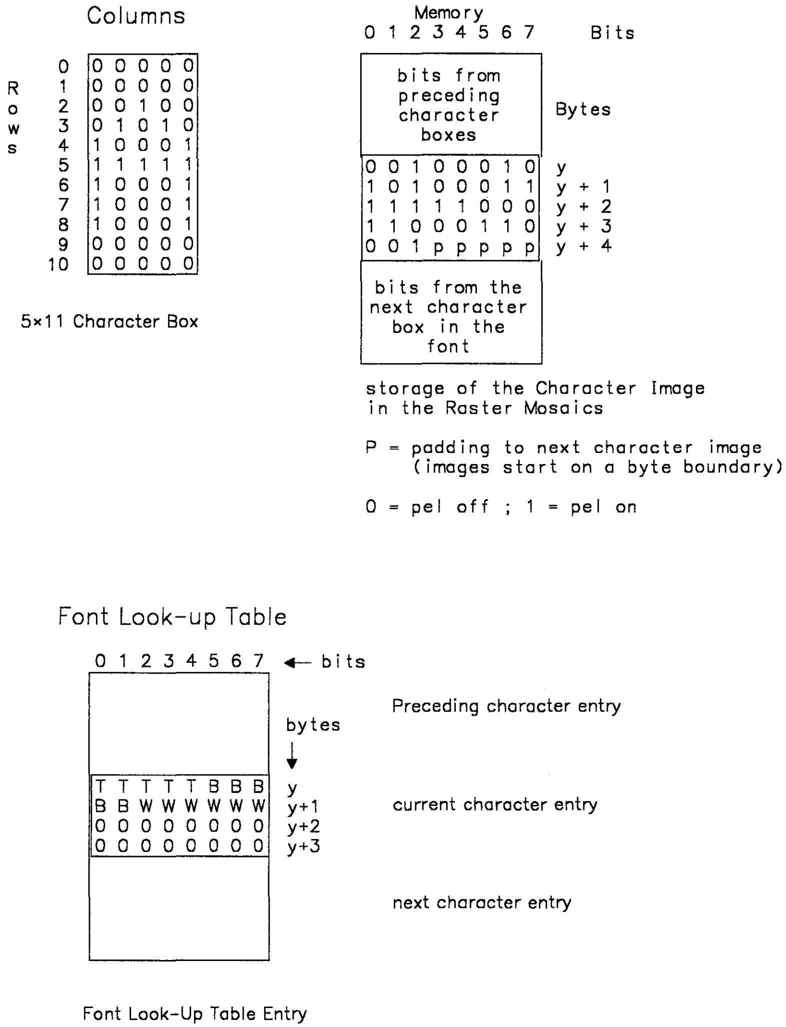


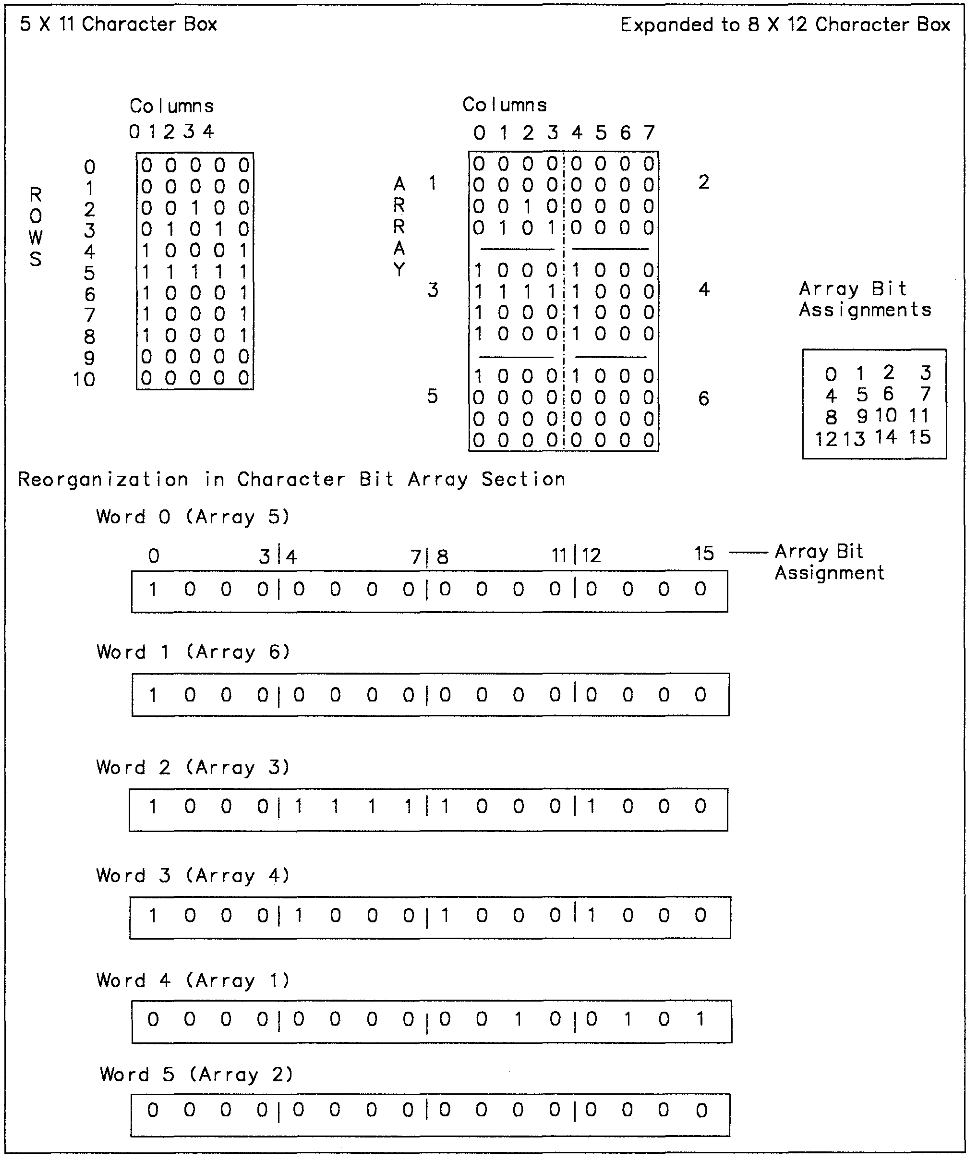
Figure 4-1. Example of Annotated Text Font Storage (non5081)

If this font is defined in a file named /usr/lib/vtm/nrml.9x20s, then compile it and convert the **a.out** file to TOC format using the following commands:

```
vcc     /usr/lib/vtm/nrml.9x20.s  -o nrml.9x20.0
vrmfmt  nrml.9x20.0  nrml.9x20
```

Annotated Text Example Two (5081)

See Figure 4-2 on page 4-72.3 for this example. The character chosen is a capital **A**, and is shown as a 5x11 character box which is then padded with zeros to the right and bottom to make the rows and columns a multiple of 4. The 4x4 arrays are then stored in 16-bit words beginning at the bottom left array in the box and continuing horizontally to the top right array in the character.



```
5 X 11 Character Box                                Expanded to 8 X 12 Character Box

            Columns                    Columns
            0 1 2 3 4                  0 1 2 3 4 5 6 7
        0   0 0 0 0 0                  0 0 0 0 | 0 0 0 0
   R    1   0 0 0 0 0       A   1      0 0 0 0 | 0 0 0 0    2
   O    2   0 0 1 0 0       R          0 0 1 0 | 0 0 0 0
   W    3   0 1 0 1 0       R          0 1 0 1 | 0 0 0 0
   S    4   1 0 0 0 1       A          -------   -------
        5   1 1 1 1 1       Y          1 0 0 0 | 1 0 0 0
        6   1 0 0 0 1           3      1 1 1 1 | 1 0 0 0    4     Array Bit
        7   1 0 0 0 1                  1 0 0 0 | 1 0 0 0          Assignments
        8   1 0 0 0 1                  1 0 0 0 | 1 0 0 0
        9   0 0 0 0 0                  -------   -------          0  1  2  3
       10   0 0 0 0 0                  1 0 0 0 | 1 0 0 0          4  5  6  7
                                5      0 0 0 0 | 0 0 0 0    6     8  9 10 11
                                       0 0 0 0 | 0 0 0 0          12 13 14 15
                                       0 0 0 0 | 0 0 0 0

Reorganization in Character Bit Array Section
     Word 0 (Array 5)
        0        3|4        7|8        11|12       15 —— Array Bit
        1  0  0  0| 0  0  0  0| 0  0  0  0| 0  0  0  0    Assignment

     Word 1 (Array 6)
        1  0  0  0| 0  0  0  0| 0  0  0  0| 0  0  0  0

     Word 2 (Array 3)
        1  0  0  0| 1  1  1  1| 1  0  0  0| 1  0  0  0

     Word 3 (Array 4)
        1  0  0  0| 1  0  0  0| 1  0  0  0| 1  0  0  0

     Word 4 (Array 1)
        0  0  0  0| 0  0  0  0| 0  0  1  0| 0  1  0  1

     Word 5 (Array 2)
        0  0  0  0| 0  0  0  0| 0  0  0  0| 0  0  0  0
```

Figure 4-2. Example of Annotated Text Font Storage (5081)

Annotated Text Font Files (non5081)

/etc/vtm/nrm1.9x20	Normal 9 by 20 font, compiled
/etc/vtm/bld1.9x20	Bold 9 by 20 font, compiled
/etc/vtm/itl1.9x20	Italic 9 by 20 font, compiled
/etc/vtm/nrm1.8x14	Normal 8 by 14 font, compiled
/etc/vtm/nrm1.4x8	Normal 4 by 8 microfont font, compiled
/etc/vtm/nrm1.18x40	Normal 18 by 40 title font font, compiled
/etc/vtm/nrm1.12x30	Normal 12 by 30 font, compiled
/etc/vtm/erg1.9x20	Ergonomic 9 by 20 font, compiled
/usr/lib/vtm/nrm1.9x20.s	Normal 9 by 20 font, source
/usr/lib/vtm/bld1.9x20.s	Bold 9 by 20 font, source
/usr/lib/vtm/itl1.9x20.s	Italic 9 by 20 font, source
/usr/lib/vtm/nrm1.8x14.s	Normal 8 by 14 font, source
/usr/lib/vtm/nrm1.4x8.s	Normal 4 by 8 microfont font, source
/usr/lib/vtm/nrm1.18x40.s	Normal 18 by 40 title font font, source
/usr/lib/vtm/nrm1.12x30.s	Normal 12 by 30 font, source
/usr/lib/vtm/erg1.9x20.s	Ergonomic 9 by 20 font, source.

Annotated Text Font Files (5081)

/etc/vtm/nrmMP1.9x20	Normal 9 by 20 font, compiled
/etc/vtm/bldMP1.9x20	Bold 9 by 20 font, compiled
/etc/vtm/itlMP1.9x20	Italic 9 by 20 font, compiled
/etc/vtm/nrmMP1.8x14	Normal 8 by 14 font, compiled
/etc/vtm/nrmMP1.4x8	Normal 4 by 8 microfont font, compiled
/etc/vtm/nrmMP1.18x40	Normal 18 by 40 title font font, compiled
/etc/vtm/nrmMP1.12x30	Normal 12 by 30 font, compiled
/etc/vtm/ergMP1.9x20	Ergonomic 9 by 20 font, compiled

Geometric Text Fonts

Geometric text fonts are also known as programmable character set (PCS) fonts and they can be used on all GSL supported devices including the IBM 5081 Display. Each character is defined as a series of moves or draws that define the shape of the character. The moves and draws are specified as X-Y pairs of signed relative values (relative to the previous ending point, or to the bottom left of the character box for the first X-Y pair). The range of the incremental values for the X and Y coordinates is -64 to +63.

Each character definition in the font consists of a 2-byte length field for the character definition followed by 2-byte X-Y entries:

```
     Length of definition        2 bytes
     sXXXXXX 1  sYYYYYY b        2 bytes
     sXXXXXX 1  sYYYYYY b        2 bytes
     sXXXXXX 1  sYYYYYY b        2 bytes
     sXXXXXX 1  sYYYYYY b        2 bytes
       .     .      .   .
       .     .      .   .
       .     .      .   .
       .     .      .   .
     sXXXXXX 1  sYYYYYY b        2 bytes

s is the sign bit (0 = positive, 1 = negative).  Negative values are
in twos complement notation.

b is the blanking bit.  If b = 1, the primitive is blanked causing
movement without display.

1 is the low order bit of the X coordinate field and must always be a 1.
```

If the first X-Y pair is a draw rather than a move, the line is drawn from the bottom left corner of the character box. A move is specified by the low-order bit of the Y coordinate being on. A draw is specified by the low-order bit being off. The last X-Y pair in the series for the character is defined by the length field.

Geometric Text Font Definition File

The PCS font definition file consists of:

- A header that contains identifier and control information
- A table of index values used to find each character definition
- The character definitions.

Offset in Bytes	Length in Bytes	Field	Description
0x00	2	length	The length of the PCS descriptor record including the length field.
0x02	4	Reserved	0x00000000
0x06	1		Bit 1 = 0 - EBCDIC = 1 - ASCII Bits 1-2 = Reserved Bits 3-7 = (Type) specifies the data format definition for programmable characters. One is defined: '00001'B = Type 1
0x07	1		Reserved (must be zero)
0x08	2	fontid	This field identifies the programmable character set. Font IDs within the range of 1025 to 3267 are reserved for one-byte character sets. Ids within the range of 32768 to 65535 are reserved for two-byte character sets.
0x0A	1	segmentid	For two-byte character sets, this byte contains the first byte of the 2-byte character code.
0x0B	1		Reserved (must be zero)
0x0C	2	P	Range of X (between 0 and P)
0x0E	2	Q	Range of Y (between 0 and Q)
0x10	1	CP0	Starting character code within PCS (within the range of 0x21 to 0xFE.)
0x11	1	CPn	The last character code within this PCS. If CPn is zero, 0xFE is assumed. CPn must not be less than CP0.
0x12	2	font baseline	The value of the font baseline in pixels in the Y direction from the bottom line of the character. This value is used in conjunction with the text alignment function.

Offset in Bytes	Length in Bytes	Field	Description
0x14	2	font capline	The value of the font capline in pixels in the Y direction from the bottom line of the character. This value is used in conjunction with the text alignment function.
0x16	1	Reserved	
0x17	1	Default error code point	This is the character code within this PCS font that specifies the character to be displayed when an invalid code is encountered or the character code does not exist.
0x24	var	Character Index	This field contains two-byte offsets to each character description. Each offset is from the beginning of the descriptor record.
var	var	Character Description	This field contains the character definitions beginning with code point CP0, in ascending order.

P and Q together define the character box within which a normal character will fit. The values of P and Q are defined in device coordinate space (pixels) and control spacing between characters and new line spacing. The bottom left corner of the box is 0,0 and the top right corner is P,Q. Characters can extend outside this box as P and Q control only the intercharacter spacing. You can override the value of P specified in the header by specifying a character inline spacing value greater than zero. Undefined character codes (outside the range CP0-CPn, or those with an index value of zero) are displayed as the default code point character.

Each character index value is the offset from the start of the header record to the actual character definition. The index must always be represented in its entirety, even if not all of the characters in the code range are defined. For example, the maximum length of the index, if CP0 is specified as 0x41 and CPn as 0xFF, is 191 times 2 bytes. For undefined characters, the index value should contain an offset to the default code definition.

Each character definition begins with a 2-byte length field which specifies the length of the character definition including the length field.

Related Information

In this book: "master" on page 4-98, "data stream" on page 5-5, "display symbols" on page 5-24, "Reconfigure (HFRCONF)" on page 6-31, "gsgtat" on page 7-73, "gsgtxt" on page 7-78, "gstatt" on page 7-128, and "gstext" on page 7-132.

The **display** command in *AIX Operating System Commands Reference.*

The discussion of the TOC object module format in *Virtual Resource Manager Technical Reference.*

fs

Purpose

Contains the format of a file system volume.

Synopsis

#include <sys/types.h>
#include <sys/param.h>
#include <sys/filsys.h>

Description

A file system storage volume has a common format for certain vital information. A volume is divided into a number of logical blocks, 512-byte blocks for diskette and 2048-byte blocks for disks. The term ***block*** here refers to the unit of disk space allocation, which is some multiple of 512 bytes. The 512-byte unit is used to report or specify file sizes in all commands and subroutines, but here the term refers to a cluster of one or more such units. RT PC supports two similar but distinct file system formats, both of which are described by the following text.

The first format uses the byte order and integer size of the native processing unit. The second format is compatible with the PC/IX file system format, which is based on the IBM PC-XT processing unit architecture. There are several differences between the two file system formats: the block size is 2048 bytes in the native file system format and 512 bytes in the other, and the definition of the superblock is different between the two. The number and order of bytes within multi-byte data are different, and the processing units impose different restrictions on the alignment of 4-byte data. These last two differences affect fields within the superblock, i-node numbers within directories, and logical block numbers within i-nodes and indirect blocks.

The **mkfs** application makes a file system of the second format only when the file system device is a diskette.

Logical block 0 is unused and available to contain a bootstrap program or other information. Logical block 1 is the superblock.

The format of a native-format file system superblock follows.

```
#define      FSfixsz    112    / * Fixed-format region is 112 bytes long */

typedef      struct     filsys
{   /* basic file system parameters - initialized when FS is created */
      char     s_magic[4];    /* magic number: FSmagic = 0xdf817eb2 */
      char     s_flag[4];     /* flag word (see below)    */
      daddr_t  s_fsize;       /* size in blocks of entire file system */
      ushort   s_bsize;       /* block size (in bytes) for this filsys */
      ushort   s_isize;       /* size (in blocks) of i-list and overhead */
      short    s_cyl;         /* number of blocks per cylinder */
      short    s_skip;        /* block interleaving factor */
      short    s_nicfree;     /* number of slots in block free list */
      short    s_nicino;      /* number of slots in free i-node list */
      short    s_sicfree;     /* byte offset to start of block free list */
      short    s_sicino;      /* byte offset to start of free i-node list */
      char     s_fname[6];    /* name of this file system */
      char     s_fpack[6];    /* name of this volume    */
      short    s_nicfrag;     /* number of slots in fragment table  */
      short    s_sicfrag;     /* byte offset to start of fragment table  */
      daddr_t  s_swaplo;      /* start of swap area (currently unused) */
      daddr_t  s_nswap;       /* number of block swap area (currently unused) */
      char     s_rsvd[36];    /* reserved - must be zero */

      /* current file system state information, values change over time */

      ushort   s_tffrag;      /* number of fragmented files (currently unused) */
      ushort   s_tbfrag;      /* number of fragmented blocks (currently unused) */
      short    s_findex;      /* fragment allocation index (currently unused */
      char     s_fmod;        /* superblock modified flag */
      char     s_ronly;       /* mounted read-only flag */
      daddr_t  s_tfree;       /* total free blocks */
      char     s_flock;       /* lock during free list manipulation */
      char     s_ilock;       /* lock during i-list manipulation */
      short    s_nfree;       /* number of addresses in s_free */
      ino_t    s_tinode;      /* total free inodes */
      short    s_ninode;      /* number of inodes in s_inode */
      time_t   s_time;        /* time of last superblock update */
```

```
       /*
        * TOTAL LENGTH OF FIXED-FORMAT REGION: 112 bytes
        *
        * All variable length fields appear beyond this point, and are
        *       described and pointed to by information in the fixed format
        *       portion of the superblock:
        *
        *       daddr_t    s_free[s nicfree];<free block list>
        *       ino_t      s_inode[s nicino];<free I-node list>
        *       frag_t     s_frag[s_nicfrag];<fragment table>
        *
        *       Macros defined below allow access to these tables.
        */

       union                        /* Variable-format */
       { char    su_var [BSIZE-FSfixsz];
         struct
                 {         daddr_t su_free[NICFREE];
                           ino_t   su_inode[NICINOD];
                 } su_ovly;
      } s_u;
} filsys_t;

#define s_free  s_u.su_ovly.su_free
#define s_inode s_u.su_ovly.su_inode
#define s_var   s_var

#define s_n     s_cyl           /* for compatibility with old systems */
#define s_m     s_skip
#define FSmagic "\337\201\176\262"  /* octal magic number for file systems */
/* hex equivalent value for FSmagic number above is \df\81\7e\b2 */
#define s_cpu   s_flag[0]   /* Target cpu type code (same as in a.out files) */
#define s_type  s_flag[3]   /* File system type code (block size) */

#define Fs1b  1          /* 512 byte blocksize file system */
#define Fs2b  2          /* 1024 byte blocksize */
#define Fs4b  3          /* 2048 byte blocksize */
#define Fs8b  4          /* 4096 byte blocksize */
```

```
/*
 *  Notes on s_fmod field:
 *  This field is intended to be a three state flag with the third
 *  state being a sticky state.  The three states are:
 *
 *  0 = file system is clean and unmounted
 *  1 = file system is mounted
 *  2 = file system was mounted when dirty
 *
 *  If you merely mount and unmount the filesystem, the flag
 *  toggles back and forth between states 0 and 1.  If you ever
 *  mount the filesystem while the flag is in state 1 then it
 *  goes to state 2 and stays there until you run fsck.
 *  The only way to clean up a corrupted file system (and change
 *  the flag from state 2 back to state 0) is to run fsck.
 *  The bit above this tri-state (i.e. 04, FM_SDIRTY) is only used
 *  in memory.  It is never written to disk.
 */
#define FM_CLEAN   00  /* File system is clean and unmounted          */
#define FM_MOUNT   01  /* File system is mounted cleanly              */
#define FM_MDIRTY  02  /* File system was dirty when last mounted     */
#define FM_SDIRTY  04  /* Superblock is dirty; this bit is not written */

#define FMOD(x)    ((x)==0?1:2)
#define FCLEAN(x)  ((x)==2?2:0)

/*
 * Macros for accessing elements in the variable-format region of the
 *  superblock. "sbp" is a pointer to superblock, and "n" gives
 *  the index of the element to be fetched.
 */

/*  FREEino() -- Finds the nth element in the free I-node list.  */
/*  Each element of the free I-node list is of type "ino_t".     */

#define FREEino(sbp,n) \
    (((ino_t *)((char *)(sbp)+(sbp)->s_sicino))[n])
```

```
/*  FREEblk() -- Finds the nth element in the free block list.  Each */
/*         element of the free block list is of type "daddr_t".  */

#define FREEblk(sbp,n) \
    (((daddr_t *)((char *)(sbp)+(sbp)->s_sicfree))[n])

/* we have a NEW Format superblock */
#define    _s_NEWF
```

The format of a PC/IX-format file system superblock is:

```
/*
 *  Structure of the superblock
 */
struct      filsys
{
   ushort     s_isize;           /* size in blocks of i-list */
   daddr_t    s_fsize;           /* size in blocks of entire volume */
   short      s_nfree;           /* number of address in s_free */
   daddr_t    s_free[NICFREE];   /* free block list */
   short      s_ninode;          /* number of inodes in s_inode */
   ino_t      s_inode[NICINOD];  /* free I-node list */
   char       s_flock;           /* lock during free list manipulation */
   char       s_ilock;           /* lock during i-list manipulation */
   char       s_fmod;            /* superblock modified flag */
   char       s_ronly;           /* mounted read-only flag */
   time_t     s_time;            /* last superblock update */
   short      s_dinfo[4];        /* device information */
   daddr_t    s_tfree;           /* total free blocks */
   ino_t      s_tinode;          /* total free i-nodes */
   char       s_fname[6];        /* file system name */
   char       s_fpack[6];        /* file system pack name */
   long       s_fill[13];        /* fill out to 512 bytes */
   daddr_t    s_swaplo;          /* start of swap area      */
   daddr_t    s_nswap;           /* number of blocks of swap */
   long       s_magic;           /* magic number for file systems */
   long       s_type             /* file system type - cluster size */
};
```

```
/*
 * macros to give more meaningful names to dinfo fields
 */
#define s_m      s_dinfo[0]       /* modulo factor in superblock */
#define s_n      s_dinfo[1]       /* cylinder size in superblock */
#define s_bsize  s_dinfo[2]       /* block size for this file system */
```

If the latter superblock structure is compiled into a program, the native compiler adds pad bytes to force **long** type data to be aligned on an address that is a multiple of 4. A program that attempts to manipulate the PC/IX format superblock must redeclare the values in a manner that does not change the given alignment and then change data references appropriately.

The parameters **NICFREE** and **NICINOD**, the number of in-core free blocks and free i-nodes, respectively, are defined in the system include file, **<sys/param.h>**, as are **BSIZE** (the number of bytes in a block), and **DIRSIZ** (the number of bytes in a simple file name).

The **s_isize** field is the number of the first data block after the i-list; the starts just after the superblock (in block 2); thus the i-list is **s_isize** minus 2 blocks long. The **s_fsize** field is the total number of blocks in the file system. These numbers are used by the system to check for bad block numbers; if an block number that cannot exist is allocated from the free list or is freed, a message is sent to the system console. Moreover, the free array is cleared, to prevent further allocation from a presumably corrupted free list.

The **s_bsize** field contains the number of bytes in a file system block.

The **s_cyl** and **s _skip** fields contain parameters that control the organization of the free-block list. The **s_cyl** field contains the number of blocks per cylinder; **s_skip** is the interleave factor. Free-list interleaving is described by the **mkfs** application. In the PC/IX format file system, these fields are referenced using macros called **s_n** and **s_m**, respectively.

The **s_nicfree** and **s_nicino** fields contain the values of **NICFREE** and **NICINO** (sizes of the **s_free** and **s_inode** arrays). The **s_sicfree** and **s_sincino** fields contain the byte offset from the start of the superblock of **s_free** and **s_inode** arrays. These numbers are provided to facilitate the writing of **BSIZE** independent file system management utilities. These fields are present only in the native format file system.

The free list for each volume is maintained as follows. The **s_free** array contains, in **s_free**[1], . . . , **s_free[s nfree**-1], the block numbers of up to **NICFREE**-1 free-blocks. The **s_free**[0] value is the block number of the head of chain of blocks constituting the free list. The first long in each free-chain block is the number (up to **NICFREE**) of free-block numbers listed in the next **NICFREE** longs of this chain member. The first of these block numbers is the link to the next member of the chain. To allocate a block: decrement **s_nfree**, and the new block is **s_free [s_nfree]**. If the new block number is 0, there are no blocks left. This an error condition. If **s_nfree** became 0, read the block named by the new block number, replace **s_nfree** by its first word, and copy the block numbers in the next **NICFREE** longs into the **s_free** array. To free a block, check whether **s_nfree** is **NICFREE**; if so, copy **s_nfree** and the **s_free** array into it, write it out, and set **s_nfree** to 0. In any event, set **s_free[s_nfree]** to the freed block's number and increment **s_nfree.**

The value of **s_tfree** is the total free-blocks available in the file system.

The value **s_ninode** is the number of free i-numbers in the **s_inode** array. To allocate an i-node: if **s_ninode** is greater than 0, decrement it and return **s_inode[s_ninode]**. If it was 0, read the i-list and place the numbers of up to **NICINOD** free i-nodes into the **s_inode** array, then try again. To free an i-node, provided **s_ninode** is less than **NICINOD**, place its number into **[s_ninode]** and increment **s_ninode**. If **s_ninode** is already **NICINOD**, do not bother to enter the freed i-node into any table. This list of in-nodes serves only to speed up the allocation process. The i-node itself indicates whether it is free.

The value of **s_tinode** is the total number of free i-nodes available in the file system.

The **s_fmod** field is a flag to indicate the "cleanliness" of the file system. A value of 0 indicates that the file system has been cleanly unmounted. Whenever a file system is mounted, this flag is checked and a warning message is printed if the **s_fmod** flag is non-zero. When a clean file system is mounted, the **s_fmod** flag is changed to a value of 1. When an unclean file system is mounted, **s_fmod** is set to 2. When a file system is unmounted, the **s_fmod** flag is reset to 0 only if it has the value 1. Thus, a file system whose **s_fmod** flag is 0 is very likely to be clean, and a file system whose **s_fmod** flag is 2 is likely to have problems.

The **s_ronly** field is a flag indicating that the file system has been mounted read only. This flag is maintained in memory only, its value on disk is not valid.

The value of **s_time** is the last time the superblock of the file system was changed, (in seconds since 00:00 Jan. 1, 1970 (GMT)).

s_fname is the name of the file system and **s_fpack** is the name of the device on which it resides.

The **s_flock** and **s_ilock** flags are maintained in the copy of the file system in memory while it is mounted; their values on disk are not valid.

The **s_fill, s_swaplo**, and **s_nswap** fields are not used on this system.

I-numbers begin at 1, and the storage for i-node 1 begins in the first byte of block 2. I-node 1 is reserved for a file without a name. This i-node is used by the **mkfs** application to put the numbers of defective blocks (blocks with physical flaws) to prevent them from being allocated to other files. I-node 2 is reserved for the root directory of the file system. No other i-number has a built-in meaning. I-nodes are 64 bytes long, so **BSIZE** ÷ 64 of them fit into a block. Each i-node represents one file. For the format of an i-node and its flags, see "inode" on page 4-92.

Files

/usr/include/sys/filsys.h
/usr/include/sys/stat.h

Related Information

In this book: "inode" on page 4-92 and "param.h" on page 5-68.

The **fsck, fsdb,** and **mkfs** programs in *AIX Operating System Commands Reference.*

fspec

Purpose

Specifies formatting within text files.

Description

A text file format specification normally occurs in the first line of a text file. This format specifies how tabs expand in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and enclosed by the brackets <: and :>. Each parameter consists of a key-letter, possibly followed immediately by a value. The following parameters are recognized:

d The **d** parameter takes no additional value. It indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no additional value. It indicates that the current format prevails until another format specification is encountered in the file.

m*margin* The **m** parameter specifies a number of spaces added to the beginning of each line. The value of *margin* must be an integer.

s*size* The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs are expanded, but before inserting the margin.

t*tabs* The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:

- A list of column numbers separated by commas, indicating tabs set at the specified columns.
- A − (dash) followed immediately by an integer *n*, indicating tabs at intervals of *n* columns.
- A − (dash) followed by the name of a supplied tab specification.

Standard tabs are specified by **t-8**, or the equivalent **t1, 9, 17, 25,** and so on. The **tabs** command defines the supplied tabs.

Default values assumed for parameters not supplied are **t-8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file contains no format specification, the previous defaults are assumed for the entire file.

The format specification can be entered as a comment. In that case it is not necessary to code the **d** parameter.

Example

The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

Related Information

The **ed, newform,** and **tabs** commands in *AIX Operating System Commands Reference.*

gps

Purpose

Used as the format for storing graphics file data as graphic primitive strings.

Description

A ***GPS*** is a graphic primitive string that is used to store graphical data in a particular format. The **plot** and **vtoc** commands produce GPS output files. Several commands edit and display GPS files on various devices. A GPS is composed of as many as five types of graphical data or primitives:

comment A *comment* is an integer string included within a GPS file that does not cause anything to be displayed. All GPS files begin with a *comment* of zero length.

lines A *lines* primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a ***move*** to that location, relocating the graphics cursor without drawing. Successive points produce line segments from the previous point.

arc An *arc* primitive has a variable number of points to which a curve is fit. The first point produces a move to that point. If only two points are given, a line connecting the points is the result. If three points are given, a circular arc through the points is drawn. If more than three points are given, splines are fitted to connect the points.

text The *text* primitive draws characters beginning at a given point, with the first character centered on that point.

hardware The *hardware* primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the *hardware* string.

Graphic primitive strings are given as 16-bit units called ***command words***. The first command word determines the primitive type and sets the length of the string. Subsequent command words contain information in multiples of ***quid***, four bits of data. The following are the types of GPS and their parameters:

comment *cw* [*string*]

cw is the control word. The first quid identifies the *comment* primitive and has the value 0xF. The following bits give the command word count for the primitive.

[*string*] is a string of characters terminated by a null character. If the string does not end on a command word boundary, another null character is added to align the string with the command word boundary.

lines *cw points sw*

cw is the control word. The first quid identifies the *lines* primitive and has the value 0x0. The remaining bits give the command word count for the primitive.

points is one or more pairs of integer coordinates having values within a Cartesian plane or universe of 65,536 points on each axis (-32,767 to +32,768).

sw is the style command word. The first eight bits hold an integer value for *color* information. The next quid contains an integer value for *weight* to indicate line thickness:

0	Narrow
1	Bold
2	Medium.

The last quid of *sw* is an integer value giving line *style* information:

0	Solid
1	Dotted
2	Dot−dashed
3	Dashed
4	Long dashed.

arc *cw points sw*

cw is the control word. The first quid identifies the *arc* primitive and has the value 0x3. The next twelve bits contain the command word count for the primitive.

points is one or more pairs of integer coordinates having values within a Cartesian plane or universe of 65,536 points on each axis (-32,767 to +32,768).

sw is the style command word. The first eight bits are an integer value for *color*. The next quid contains an integer value for *weight* to indicate line thickness:

0	Narrow
1	Bold
2	Medium.

The last quid is an integer value setting line style:

0	Solid
1	Dotted
2	Dot−dashed

3 Dashed
4 Long dashed.

text *cw point fw so* [*string*]

cw is the control word. The first quid identifies the *text* primitive and has the value 0x2. The remaining twelve bits contain the command word count for the primitive.

point is a pair of integer coordinates that are a value within a Cartesian plane or universe of 65,536 points per axis (-32,767 to +32,768).

fw is a font command word. The first eight bits contain an integer value for *color* information. The next eight bits contain an integer value for *font* information, with a quid qiving a *weight* (density) value for the font, and a quid giving a *style* (typeface) value for the font.

so is a size/orientation command word. Eight bits specify *textsize* as an integer value to indicate the size of characters drawn. *textsize* represents character height in absolute *universe units.* The actual character height is five times the *textsize* value. The next eight bits are a signed integer value for *textangle,* and express the angle and direction of rotation of the character string around the beginning *point. textangle* is expressed in degrees from the positive $x-$axis. The *textangle* value is 256/360 of its absolute value.

hardware *cw point* [*string*]

cw is the control word. The first quid identifies the *hardware* primitive and has the value 0x4. The next twelve bits indicate the command word count for the primitive.

point is a pair of integer coordinates that are values within a Cartesian plane or universe of 65,536 points on each axis (-32,767 to +32,768). This *point* is the starting point for the *string,* which is a string of hardware characters or control commands to a hardware device.

Related Information

In this book: "stat.h" on page 5-69.

The **stat** and **toc** commands in *AIX Operating System Commands Reference.*

group

Purpose

Identifies a group.

Description

Users can be assigned to one or more groups, each of which share certain protection privileges. The person who sets up the system may want to place users in the same group because they need access to a common set of files. Similarly, a certain group of users can have access restricted to certain files.

When users log in, they are assigned to the group specified in the **password** file. In addition, they are assigned as a member of all groups specified in this file. Users are allowed to access to any files that the group to which they are assigned has access. However, any files created by the user can be accessed only by the members of the primary group of which that user is a member. A user is allowed to change his primary group for the duration of the terminal session using the **newgrp** command.

The **group** file defines to which groups a user has membership. Each line in this file defines a group and consists of four fields separated by colons. It contains the following information for each group:

group name — A character string of up to 8 characters that references the group.

password — This field is optional. If specified, anyone attempting to enter the group must correctly supply the password to the system.

group ID — A number assigned to the group and used in access decisions.

user group list — A list that specifies the login names of all users allowed in the group. User IDs in the list are separated by commas.

In newly distributed systems, there are typically only two groups: the staff group and the system group. New users can be added to groups and new groups can be added as necessary.

If several users wish to share the same privileges, including the ability to terminate each other's processes as well as to access the files of others, the same numerical user ID can be assigned to each. This mechanism is sometimes used to give the same person several accounts on the system, each with potentially different login directories and other characteristics, such as electronic mailboxes or login programs. For example, the operator has the same user ID, and therefore superuser authority. However, this operator typically uses a restricted version of the shell that does not give access to commands that allow reading the files of others.

Example

The following is an example of a group file. This is an ASCII file. Each group is separated from the next by a new-line character. The fields are separated by colons. This file resides in **/etc/group**. Because the password is encrypted, it can be used to map numerical group IDs to names without concern of compromise to user security.

```
system::0:su,bill,jack,gary
staff::1:
bin::2:su,bin
sys::3:su,bin.sys
adm::4:su,bin,adm
mail::6:su
usr::100:guest
```

File

/etc/group

Related Information

In this book: "passwd" on page 4-112.

The **newgrp, passwd,** and **users** commands in *AIX Operating System Commands Reference.*

history

Purpose

Contains the history of an installed licensed program product.

Description

Each licensed program or component of a licensed program that is shipped by IBM contains a history file. The purpose of a history file is to identify the installed release and version of a licensed program or component and to provide a record of any updates (level changes). A history file is replaced when a component is reinstalled. History files for programs installed on the operating system are named **/usr/lpp/***pgm-name***/lpp.hist**, where *pgm-name* is the name of the licensed program or component. History files for programs installed completely on the VRM minidisk are named **/vrm/lpp/***pgm-name***/lpp.hist**

The history file consists of a series of 80-character records. The first 2 records contain the install data and all subsequent records contain update data. There are 3 different formats of 80-character records:

Record	**Description**
Information	Identified by an **a**, **c**, **r**, or **v** character in position 1. The **install** and **update** procedures use information records to identify the licensed program or component name; the current version, release, and level; the date the record was added; and the user who initiated the install or update. Figure 4-2 on page 4-90 shows the format of the fields in the information records.
Title	Identified by a **t** in character position 1. Contains the descriptive title (up to 30 characters) for the licensed program or component, starting in character position 3. The title record must always be the second record in the history file.
Comment	Identified by an ***** (asterisk) in character position 1. Allows descriptive comments to be entered into the history file. An ***** is usually placed in character position 79 to ensure a full 80-column record.

The last character of each record (character position 80) must be a new-line character. Unused character positions must be blank-filled. Tab characters are not permitted.

The first record in a history file must be an information record with a **c** in character position 1. The second record must be the title record. These two records contain data about the installation of the program. The remaining records in the file may be any combination of information and comment records, and they identify updates to the program.

Figure 4-2 shows the format of an information record in the history file. The definitions for each of the fields other than character position 1 are explained following the figure.

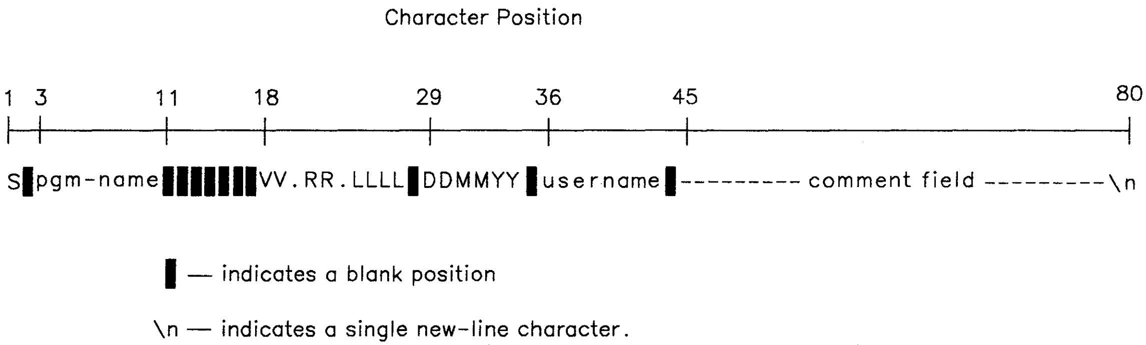

Character Position
1
3
11
18
29
36
45
80
S▮pgm-name▮▮▮▮▮▮▮VV.RR.LLLL▮DDMMYY▮username▮--------- comment field ---------\n
▮ — indicates a blank position
\n — indicates a single new-line character.

Figure 4-2. Information Record Format

Field **Description**

S The type of information record:

- **a** Indicates that the update has been applied.
- **c** Indicates that the update or install has been committed (accepted).
- **r** Indicates that the update has been rejected.
- **v** Indicates that the VRM minidisk has been modified.

pgm-name The name assigned to the program (lowercase characters only). If the name contains less than 8 characters, it must be padded with blanks.

VV. A 2-digit numeric field followed by a period indicating the version number of the program. The version number indicates the level of the hardware and operating system with which the program works.

RR. A 2-digit numeric field followed by a period indicating the release number of the program. The release number tracks changes to external programming interfaces since the last version change. This number is generally incremented each time the external interface to the program changes.

LLLL A 4-digit numeric field indicating the update level of the program. This field is incremented when the changes to the program do not affect external programs that may use the documented external interface for the program. The level, together with the *S* field, ensures that all changes up to and including the current change are installed on the system.

The fourth (or units) digit of the level is normally 0. IBM reserves this digit for future use.

DDMMYY	These three numeric fields indicate the date the program changed: *DD* Day of the month (1 to 31). *MM* Month of the year (1 to 12). *YY* Year (00 to 99).
username	An alphanumeric field that contains the user name of the person who installed the program. If the user name is shorter than eight characters, it must be padded with blanks.
comment field	A 35-character field for comments. An * (asterisk) is usually placed in character position 79 to ensure a full 80-column record.
\n	A required new-line character, indicating the end of the record.

Files

/usr/lpp/*pgm-name***/lpp.hist**
/vrm/lpp/*pgm-name***/lpp.hist**

Related Information

The **installp** and **updatep** commands in *AIX Operating System Commands Reference.*

inode

Purpose

Describes a file system file or directory entry as it appears on a disk.

Synopsis

#include <sys/types.h>
#include <sys/ino.h>

Description

An **inode** for an ordinary file or directory in a file system has the following structure defined by **sys/ino.h**:

```
 /* Inode structure as it appears on a disk block. */
struct dinode
{
      ushort  di_mode;    /* mode and type of file */
      short  di_nlink;    /* number of links to file */
      ushort  di_uid;     /* owner's user id */
      ushort  di_gid;     /* owner's group id */
      off_t  di_size;     /* number of bytes in file */
      char   di_addr[40]; /* disk block addresses */
      time_t  di_atime;   /* time last accessed */
      time_t  di_mtime;   /* time last modified */
      time_t  di_ctime;   /* time created */
 };
 /*
 *the 40 address bytes:
 *   39 used; 13 addresses
 *   of 3 bytes each.
 */
```

The fields in the structure are as follows:

addr Array of thirteen 3-byte block numbers assigned to this file. The first 10 block numbers are direct addresses while the last 3 are indirect addresses.

atime	Time this file was last accessed.
ctime	Time this file was created.
gid	Group ID.
mode	Type and access permissions of file.
mtime	Time this file was last modified.
nlink	Number of directories that name this file.
size	Number of bytes in file.
uid	Owner ID.

See the **types** file for related information concerning the **off_t** and **time_t** define types.

Related Information

In this book: "fs" on page 4-74, "stat.h" on page 5-69, and "types.h" on page 5-75.

kaf

Purpose

Specifies how to process **ddi** keywords and their parameters.

Description

Keyword Attribute Files, also called **kaf** files, define how the **devices** command and customize helpers are to process keywords used in **ddi** files. The **kaf** files:

- Contain instructions for processing device information
- Control whether the **devices** command displays the associated information
- Control whether a user can change the information using the **devices** command
- Specify the input validation that the **devices** command performs
- Determine the action that the customize helper takes.

The **kaf** information can be included in the **ddi** file for the device, or it can appear in a separate file. If it is contained in a separate file, then the stanza for the device in the **system** file must name the **kaf** file as the value of the **kaf–file** keyword. The **kaf–use** keyword (also in the **system** file) specifies the stanza of the **kaf** file to use.

The name of each stanza in a **kaf** file is the name of a keyword that is used in **ddi** files. The stanza defines how the **devices** command and customize helper programs process that **ddi** keyword. The following section defines the keywords that can appear in the stanzas of **kaf** files.

The use of extended characters in **kaf** files is not supported.

Directives to the Customize Helper

add Specifies the actions for the customize helper to take during a **vrmconfig -a** (add) operation.

delete Specifies the actions for the customize helper to take during a **vrmconfig -d** (delete) operation.

startup Specifies the actions for the customize helper to take during a **vrmconfig -startup** operation.

shutdown Specifies the actions for the customize helper to take during a **vrmconfig -shutdown** operation.

The value for each of the preceding keywords has the format *x*/*y*, where *x* and *y* can be any of the following:

b Constructs the Define_Device Structure, including the Block I/O Communication Area (BIOCA) device characteristics. Appends other device characteristics specified by the programmer. (See the discussion of customize helpers in *AIX Operating System Programming Tools and Interfaces.*) Performs the Define_Device SVC by issuing an **ioctl** system call to the **config** device driver.

n Constructs the Define_Device Structure. Appends other device characteristics specified by the programmer. (See the discussion of customize helpers in *AIX Operating System Programming Tools and Interfaces.*) Performs the Define_Device SVC by issuing an **ioctl** system call to the **config** device driver.

u Constructs the AIX device driver structure and issues an **ioctl** system call to the **config** device driver to initialize the driver.

Action to Take After Customization

syschg Specifies the action the **devices** command takes when the user changes a device characteristic. The valid choices are:

- **a** Rebuilds the kernel and IPLs the system
- **s** Runs the special processing routine specified by the **specproc** keyword in the **/etc/system** stanza.
- **none** Takes no special action.

Control over Display and Modification of the Keyword

display If set to **true**, then the **devices** command displays the device characteristic keyword and allows the user to change its value.

dsrc Determines whether the **devices** command displays the adapter characteristic keyword from the **ddi** file. The value is a list of adapter numbers, separated by commas. The **devices** command displays the keyword and allows the user to change its value only if the number of the adapter associated with the device appears in the list.

required If set to **true**, the **devices** command displays the keyword and advises the user to make sure that its value matches the system configuration. **devices** does not check to see whether the entered value matches the system configuration.

rsrc Determines whether the **devices** command displays the adapter characteristic keyword from the **ddi** file and requires the user to enter a value. The value is a list of adapter numbers, separated by commas. If the number of the adapter associated with the device appears in the list, then the **devices** command displays the keyword and requires the user to specify its value. **devices** does not check to see whether the entered value matches the system configuration.

User Input Validation

vtype Specifies the type of checking that the **devices** command performs on values entered by the user. **vtype** can be set to one of the following values:

0 No validation.
1 Mapping validation: the input value must be one of the keywords found in the stanza named by the **map** keyword.
3 Range validation: The input value must have the data type specified by the **type** keyword and must fall in the range specified by the **range** keyword.

map Names a stanza in the **kaf** file that contains a list of *keyword*=*value* pairs against which the input value is to be matched. If the input matches a given *keyword*, then the corresponding **value** is substituted in its place.

opts Specifies the *option*s to search for in the **/etc/ddi/options** file. **opts** is one of the following:

k Keyword only
a Keyword followed by adapter name
c Keyword followed by device class
t Keyword followed by device type
s Keyword followed by device stanza name.

If the **opts** keyword is not specified, its value defaults to **k**. See “options” on page 4-110 for details about the **/etc/ddi/options** file.

range Defines the valid range of values for a keyword so that the **devices** command can verify values entered by the user. The value of the **range** keyword has the format *first*,*last*,*incr*, where *first* is the first number in the range, *last* is the last number, and *incr* is the increment between values in the range. For example, `range=2,10,2` specifies the values 2, 4, 6, 8, and 10.

type Defines the data type for the value of a keyword. The **devices** comand ensures that the values entered are the correct data type, specified by one of the following:

F Floating-point (**float**)
H Hexadecimal (**int**)
I Integer (**int**)
L Long integer (**long int**)
S Short integer (**short int**)
U Unsigned integer (**unsigned int**).

Files

/etc/ddi/font.kaf
/etc/ddi/opprinter.kf
/etc/ddi/osprinter.kf
/etc/ddi/plotter.kaf
/etc/ddi/pprinter.kaf
/etc/ddi/sprinter.kaf
/etc/ddi/tty.kaf
/etc/mdkaf

Related Information

In this book: "attributes" on page 4-20 "ddi" on page 4-43, "descriptions" on page 4-56, "system" on page 4-139, and "config" on page 6-7.

The discussion of customize helpers in *AIX Operating System Programming Tools and Interfaces.*

master

Purpose

Contains master configuration information.

Description

The **master** file is an attribute file that contains stanzas that describe all device drivers defined in the system. There are two kinds of stanzas, AIX device driver stanzas and Virtual Resource Manager (VRM) driver stanzas. AIX driver stanzas specify drivers to link into the kernel and the VRM drivers that support them. VRM driver stanzas specify drivers to be loaded into the VRM at the time the system is loaded.

The use of extended characters in the **master** file is not supported.

The sysparms Stanza

The first stanza of the **/etc/master** file, the **sysparms** stanza, defines the values for many system parameters and limits. If you need to modify any of these system parameters, first make the changes in the **/etc/master** file, then rebuild the kernel. See "Rebuilding the AIX Kernel" on page C-51 for instructions on rebuilding the kernel.

callouts Specifies the number of callouts the kernel uses for event waiting.

charlists Specifies the number of character lists the terminal driver uses.

*driver***nkprocs** Specifies the maximum number of kernel processes available for a given device driver. To create this kind of keyword, replace *driver* with an abbreviation for the device driver, then follow it with the letters **nkprocs**. (For example, `tcpipnkprocs`.)

Note: A keyword ending with the letters **nkprocs** should be defined in the **/etc/master** file for any device driver that needs to allocate kernel processes.

dsnkprocs Specifies the maximum number of kernel defined processes available for use by Distributed Services.

dumpdev Species the target device for kernel dumps.

filetab Specifies the number of entries in the kernel open file table.

floating Indicates whether the kernel should attempt to use floating-point hardware, if it is present. Options in this parameter are **hardware** and **software**. The default, **software**, means there is no optional floating-point accelerator hardware.

hashbuffers Specifies the number of hash buffers the kernel uses.

hftbuffers Specifies the number of virtual terminals.

inodetab Specifies the number of entries in the kernel i-node table.

iobuffers Specifies the number of physical I/O buffers the kernel supports.

kbuffers Specifies the number of disk buffers in the kernel. If **kbuffers** is not defined, or if it is set to 0, then the system chooses the number of buffers based on the processor and the amount of real memory installed in the system.

kdmabuffers Specifies the number of buffer headers used by the **exec** system call. The default value is 32.

kmap Specifies the number of elements in resource map array for internal kernel storage.

kprocs Specifies the number of kernel defined processes the kernel supports.

connections Specifies the maximum number of concurrent network connections allowed for Distributed Services.

Note: This keyword sets the size of the node table in the kernel.

maxprocs Specifies the maximum number of processes for each terminal session.

mountab Specifies the number of entries in the mount table used by the kernel for file system mounting. This value sets the limit of the number of file systems that can be mounted by the kernel. See “mnttab” on page 4-108 for details.

msgmap Specifies the number of entries in message map array.

msgmax Specifies the maximum number of bytes allowed in a single message.

msgqid Specifies the maximum number of message queue identifiers allowed.

msgqmax Specifies the maximum number of bytes allowed on a message queue.

msgseg Specifies the number of message segments.

msgsegsize Specifies the message segment size (in multiples of word size).

msgtql Specifies the text queue length.

nflocks Specifies the maximum number of simultaneously locked file regions.

netnoone Specifies the Distributed Services user ID or group ID value to use under certain ID translation circumstances. This value is used when no user ID from the local node maps to a remote file's owner ID. The default value is 0xFFFE.

netsomeone Specifies the Distributed Services user ID or group ID value to use under certain ID translation circumstances. This value is used when more than one local ID maps to a remote file's owner ID, and it is difficult to select one particular ID (perhaps because of wild card mappings). The default value is 0xFFFF.

nid Specifies the node ID to generate into the system. This keyword is currently unused.

nncb Specifies the size of the Distributed Services translate table array. Each node for which the kernel has user ID or group ID translate information has an entry in this array of translate headers.

node Specifies the node name to generate into the system.

pinkbuffers Specifies whether or not the disk buffers should be pinned. The value can be **true** or **false**.

pipedev Names the stanza that defines the file system used for FIFO files.

power Indicates whether the kernel has power warning code. If this value is **true**, power warning code exists. The default is **false**.

procs Specifies the total number of simultaneous processes the kernel supports.

pslotkill Specifies the threshold at which the system begins to **kill** processes in order to recover paging space. **pslotkill** is specified in slots, where a ***slot*** is 2048 bytes (four blocks) of a paging minidisk. The default value is 200 slots.

pslotpanic Specifies the threshold at which to stop AIX and attempt a system dump because paging space has almost been exhausted. Note that the system dump itself cannot finish because of the lack of paging space. **pslotpanic** is specified in slots, where a ***slot*** is 2048 bytes (four blocks) of a paging minidisk. The default value is 100 slots.

pslotwarn Specifies the threshold at which the system displays a message warning that paging space is running low. When the system displays this message, it also:

- Performs a **sync** to write all changes to disk
- Enters ***sync mode***, in which disk I/O is not buffered
- Sends all processes the **SIGDANGER** signal to warn them that the system is likely to "crash" any moment.

pslotwarn is specified in slots, where a ***slot*** is 2048 bytes (four blocks) of a paging minidisk. The default value is 350 slots.

ptybuffers Specifies the number of pseudo-terminals that can be present in the system (see "pty" on page 6-107). The maximum value for **ptybuffers** is 16.

release Specifies the operating system release number to generate into the system.

rootdev Names the stanza in the **/etc/system** file that defines the root file system device.

rsbuffers Specifies the number of buffers allocated for the **asy** terminal driver.

semadjmax Specifies the maximum value allowed for semaphore adjust value on exit.

semid Specifies the number of distinct semaphore identifiers the kernel supports.

semmap Specifies the number of entries in semaphore map array.

semmax Specifies the maximum number of simultaneous semaphores allowed and supported by the kernel.

semopmax Specifies the maximum number of operations allowed for each **semop** system call.

semsetmax Specifies the maximum number of semaphores allowed in a set.

semunmax Specifies the number of semaphore undo structures the kernel supports.

semunpmax Specifies the maximum number of undo entries for each process.

semvalmax Specifies the maximum value allowed for each semaphore.

shmid Specifies the number of distinct shared memory identifiers the kernel supports.

shmmax Specifies the maximum number of kilobytes for shared memory allowed per shared segment.

shmmin Specifies the minimum number of kilobytes for shared memory allowed per shared segment.

shmsegs Specifies the number of segment registers that may be used to support shared memory.

slice Specifies the percentage of time in ***quanta*** that a process can run before it must relinquish control of the processor. Each ***quantum*** on the RT PC system is equal to 333 milliseconds.

system Specifies the system name to generate into the system.

texttab Specifies the number of shared text segment entries in the the text table.

version Specifies the version number of the operating system to generate into the system.

AIX Driver Stanzas

There is a unique set of keywords associated with each type of stanza. It is not necessary, however, for a stanza to contain all the keywords associated with that type. If a keyword is omitted from the stanza, the default is used. Mandatory keywords must be supplied and are not defaulted. The name of each stanza is a logical AIX driver name referenced in other stanzas.

The lines interpreted by the **config** and **vrmconfig** commands are:

config Indicates that this device has a customizaton helper program, which provides assistance in decoding other options. This value is the name of the helper program in the **/etc** directory. See "config" for more information about customizaton helper programs.

major Identifies the major device number for this driver. This is mandatory.

mandatory Identifies this driver to be included whether or not the **system** file asks for it. If this value is **true**, include this driver.

maxminor States the maximum number of minor devices this driver supports. This number should agree with the driver code.

mpx Identifies a multiplexed special file when this value is **true**.

prefix Provides a prefix for the driver routines. For example, if this value is abc, then the **open** routine in the driver is abc**open**. This keyword is mandatory. Note that all drivers are assumed to be archived into the system object libraries.

routines Identifies the routines actually defined for this driver. The possible routines are **open**, **close**, **read**, **write**, **strategy**, **ioctl**, **init**, and **print**.

tty Identifies whether the device is a terminal. If this value is **true**, the device is a terminal and terminal structures are defined.

vdriver Names the VRM driver stanzas for the related VRM drivers.

Other lines can be included for interpretation by customizaton helper programs.

VRM Driver Stanzas

The **iocn** lines identify VRM driver stanzas. The name of each stanza is a logical VRM driver name referenced in other stanzas.

The lines interpreted by the **vrmconfig** command are:

code Specifies the full path name of the file containing executable VRM code that contains the **table of contents** format of the VRM driver.

copy Names a previously specified VRM driver stanza to be used instead of the **code** keyword specification.

ctype Indicates the code type, such as **vdrvr**. This is an informational keyword for IBM customization helpers.

iocn Assigns the decimal I/O code number to this driver.

protocol If the value is **true**, indicates that this stanza describes a protocol procedure.

Other lines can be included for interpretation by customizaton helper programs.

Miscellaneous System Parameters

Both the **master** and the **system** file can have option lines describing miscellaneous system customizing and tuning options in the **sysparms** stanzas. Options in the **system** file override those in the **master** file. These options include:

inetlen Specifies the Internet packet length for file transfer. (See the **xftp** command in *Interface Program for use with TCP/IP.*) The default value is 1064 bytes.

level Specifies the level number of the operating system to generate into the system.

msgheader Specifies the maximum number of system message headers allowed.

Other keywords can be added as needed.

Example

The following sample of a master file entry contains AIX Operating System and VRM information.

```
* AIX drivers, identified by "major" keyword

* printer drivers

u5182mp:
        major = 6
        prefix = lp
        routines = open,close,write,ioctl,init
        maxminor = 8
        vdriver = v5182mp
        config = vrcmain

u5182spl:
        major = 6
        prefix = lp
        routines = open,close,write,ioctl,init
        maxminor = 8
        vdriver = v5182sp1
        config = vrcmain

u5182sp2:
        major = 6
        prefix = lp
        routines = open,close,write,ioctl,init
        maxminor = 8
        vdriver = v5182sp2
        config = vrcmain


* VRM driver entries

v5182mp:
        iocn = 2014
        code = /vrm/vrmdd/vpptr
        ctype = vdrvr

v5182sp1:
        iocn = 2015
        code = /vrm/vrmdd/vpptr
        ctype = vdrvr

v5182sp2:
        iocn = 2016
        code = /vrm/vrmdd/vpptr
        ctype = vdrvr
```

File

/etc/master

Related Information

In this book: "mount" on page 2-71, "vmount" on page 2-180.5, "mnttab" on page 4-108, "attributes" on page 4-20, "system" on page 4-139, and "pty" on page 6-107.

The **vrmconfig** and **config** commands in *AIX Operating System Commands Reference.*

message

Purpose

Describes message, insert, and help formats.

Synopsis

# include <msg10.h>

Description

The **puttext** command is used to convert message, text insert, and help descriptions from an format that can be edited into a format that can be accessed at run time. The descriptions in the file can be accessed by using the **msgimed, msgqued, msghelp,** and **msgrtrv** subroutines. The **gettext** command converts the descriptions back into a format that can be edited.

The file header contains a unique identifier indicating the type of file, a file format version number (currently 0), and the number of component entries in the file (currently, only one component entry per file is supported). The header file has the following form:

```
struct filehdr  {            /* FILE HEADER */
   char     unique[8];       /* unique file identifier "MSGSFILE" */
   unsigned short  version;  /* file format version number */
   unsigned short  numcomp;  /* number of component entries in file */
   };
```

Following the file header is the component index table. Each entry (currently, there is only one) in the table identifies the component, the national language (EN for English), the maximum index numbers that have been allocated and the offsets to the message index table, insert index table and help index table.

```
struct cmp_indx   {          /* Component index table entry */
  char             compid[6];  /* component ID */
  char             langid[2];  /* language ID */
  unsigned short flags;        /* reserved for flags (zero) */
  unsigned short maxnum[3];    /* max index numbers used for */
                               /* messages, inserts, and helps  */
  unsigned long offset[3];     /* offsets to msg, insert, and help */
                               /* index tables from start of file */
```

```
   unsigned long  reserved;    /* reserved */
   };
```

The component index table is followed by the message index table and message text, the insert index table and insert text, and help index table and help text. The header for each entry in the message, insert, and help index tables identifies the component ID and index number where the text actually resides, the offset to the text (and its length) if the text actually resides in this entry, the version number (used with a common file), and an indicator of whether the entry is current (can be accessed) or null.

```
               /* Format of header for entries in the    */
               /* message, insert, and help index tables */
               /* (Note that each index table must be */
               /*  aligned on a long integer boundary.) */

   #define MSGHEADR
      char            compid[6];  /* component ID for text source */
                                  /* file ('======' or 'common')  */
      unsigned short  index;      /* index # for text source (zero   */
                                  /* indicates same index # */
      unsigned long   offset;     /* offset to text from start of   */
                                  /* index table                   */
      unsigned short  textlen;    /* text length (not incl null term) */
      unsigned short  version;    /* version  */
      unsigned short  flags;      /* flag definition */
                                  /*  01 off:  status = null */
                                  /*     on:  status = current */
                                  /*    (other flags reserved (zero))  */
      unsigned short reservel;    /* reserved (zero) */

              /* flag definitions for MSGHEADR */
     #define mih_status 0x0001    /* off (0): status = null */
                                  /* on (1):  status = current */
```

Each entry in the insert index table contains only the header information.

```
    struct ins_indx   {           /* Insert table entry  */
                                  /* (contains header info. only)  */
       MSGHEADR                   /* header information */
       };
```

Each entry in the message index table and help index table contains the header information plus the title length (used for helps), a message/help manual reference, and the index number for the help associated with a message.

```
struct mih_indx  { /* Entry in message or help index tables.  May */
                   /* also be used as entry in insert index table */
                   /* if only header information is referenced. */
   MSGHEADER                      /* header information */
   unsigned short titlelen;       /* title length (not incl null term ) */
   char           dcompid[3];     /* displayed component ID */
   char           dmsgid[3];      /* displayed message help ID */
   short          helpindx;       /* help index number (zero if no help, */
                                  /*  negative if help in common file) */
   unsigned short reserved;       /* reserved (zero)  */
};
```

Each index table must be aligned on a long integer boundary.

Related Information

In this book: "msghelp" on page 3-252, "msgimed" on page 3-255, "msgqued" on page 3-259, and "msgrtrv" on page 3-263.

The **gettext** and **puttext** commands in *AIX Operating System Commands Reference.*

mnttab

Purpose

Provides a table of mounted file systems.

Synopsis

#include <mnttab.h>

Description

The **mnttab** file contains a table of devices, mounted by the **mount** command. Each **mnttab** file entry has the following structure:

```
struct mnttab {
    char mt_dev[100];
    char mt_filsys[100];
    short mt_ro_flg;
    time_t mt_time;
  };
```

The fields indicate the following:

mt_dev The null-padded name of the mounted special file.

mt_filsys The null-padded name of the mounted-on directory.

mt_ro_flg This flag indicates if the file system is mounted read only. A value other than 0 indicates the file system is mounted read only.

mt_time The time the file system was mounted.

The **mountab** parameter, which is defined while the system is being customized, determines the number of simultaneously mounted file systems and therefore the maximum number of entries in the **/etc/mnttab** file. This parameter changes when the **mountab** parameter in the **master** file changes.

Files

/etc/mnttab

Related Information

In this book: "filesystems" on page 4-64, "master" on page 4-98, and "system" on page 4-139.

The **config, mount,** and **umount** commands in *AIX Operating System Commands Reference.*

options

Purpose

Defines the valid choices for each **ddi** option.

Description

The **/etc/ddi/options** file contains a sorted list of the valid choices for each keyword used in **ddi** files. The **devices** command uses this file to display the valid choices for the keywords during the **add**, **change**, and **showdev** subcommands.

Each line must follow the following format:

optionchoices

where:

option This field is exactly 20 characters long, is padded on the right with spaces, and contains no tab characters. An *option* is one of the following:

keyword The keyword for which the valid choices are to be specified

keywordadapter The keyword followed by the adapter name

keywordclass The keyword followed by the device class

keywordtype The keyword followed by the device type

keywordstanza The keyword followed by the name of the device stanza in the **system** file.

The **devices** command looks for one of these combinations based on the setting of the **opts** keyword in the **kaf** file for the device. See "kaf" on page 4-94 for details about the **opts** keyword.

choices This field is exactly 29 characters long, is padded on the right with spaces, and contains no tab characters.

Note: The **/etc/ddi/options** file must be sorted alphabetically by the *option* field. If it is not sorted, then the **devices** command displays incorrect information about the options available for a given keyword.

The use of extended characters in the **etc/ddi/options** file is not supported.

File

/etc/ddi/options

Related Information

In this book: "ddi" on page 4-43, "descriptions" on page 4-56, and "kaf" on page 4-94.

passwd

Purpose

Contains passwords.

Library

Standard C Library (**libc.a**)

Synopsis

#include (pwd.h)

Description

The **passwd** file is an ASCII file that contains all the information that defines a user on the system. It contains the following information:

- Login name
- Encrypted password
- Numerical user ID
- Numerical group ID
- Additional data for each user
- Initial current directory
- Program to use as shell.

Each field is separated from the next by a colon. The file has general read permission and the passwords are encrypted. Therefore, a user can use the file to map numerical user IDs to names without potentially compromising the security of other users.

The **adduser** command is used to maintain this file. Programs should use the **getpwent** subroutines to extract various fields in this file.

If the user password field is null, the user has no password. If the program field is null, the shell (**/bin/sh**) is used. The program field can contain parameters passed when the **exec** system call is issued. Parameters are separated by space (such as a space or tab characters). A \ (backslash) is used for escapement when a parameter contains a space. The **login** command accepts the program name and as many as 14 parameters. Any more than 14 parameters are ignored. A maximum of 4096 characters can be used for the program name and its parameters. More than 4096 characters causes **login** to exit. Parameters in this field can use symbolic escapement for the following special characters: **\n**, **\r**, **\v** (produces 013), **\b**, **\t**, and **\f**. Additionally, **\0** through **\7** builds a one-byte octal number. Anything else that is preceded with a \ (backslash) passes through.

The contents of the additional data for each user has the following format:

full_name / *file_limit* ; *site_info*

where:

full_name Contains the name of the user whose 8-character (or fewer) login name is in the first field.

file_limit Specifies the maximum length file the user can create. See the **login** command in *AIX Operating System Commands Reference* and the **ulimit** system call.

site_info Contains any printable character other than a colon. This subfield is unused by the system software and is available for information for each user as required by applications specific to the site.

Any or all of the subfields can be omitted. If the **file_limit** subfield is omitted, the preceding / (slash) is omitted and the system-wide default limit is used. If the **site_info** subfield is omitted, the preceding **;** (semi-colon) is also omitted.

Passwords

The encrypted password is 13 characters long. The characters used come from the &ncs. (code page P0, see "data stream" on page 5-5) and may be uppercase or lowercase characters, numerals, and the **.** (period) and / (slash) characters except when the password is null. In this case, the encrypted password is also null. Password aging affects a particular user if a comma and a string of characters that are not null follows the user password in this file. Such a string must be initially introduced by a person with superuser authority.

The first character of the age, *M* for example, is the maximum number of weeks a password is valid. The next character, *m* for example, is the minimum number of weeks, before the password can be changed. The remaining characters indicate when the password was last changed, given as the number of weeks since the beginning of 1970 to the time of the password change. A null string is equivalent to 0. *M* and *m* have numerical values in the range 0 through 63. If $m = M = 0$, the user is forced to change the password at the next login. This causes the age to disappear from the password file entry. If $m > M$, only someone with superuser authority is able to change the password.

File

/etc/passwd

Related Information

In this book: "ulimit" on page 2-167, "a641, l64a" on page 3-4, "crypt, encrypt" on page 3-42, "getpwent, getpwuid, getpwnam, setpwent, endpwent" on page 3-219, "group" on page 4-87, and "data stream" on page 5-5.

The **login** and **passwd** commands in *AIX Operating System Commands Reference.*

"Overview of International Character Support" in *IBM RT PC Managing the AIX Operating System.*

plot

Purpose

Provides the graphics interface.

Description

The subroutines described in "plot" produce output files of the format outlined in this section. The **tplot** commands interpret these graphics files for various devices, performing the plotting instructions in the order that they appear.

A graphics file consists of a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. A point is designated by four bytes representing the *x* and *y* values; each value is a two-byte signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the current point for the next instruction.

The following table lists each of the **plot** instructions and the corresponding **plot** subroutines.

Instr	Sub	Description
a	**arc**	Draws the arc described by the following 12 bytes. The first four bytes describe the center point (x, y) of the arc or circle. The second four bytes describe the beginning point of the arc. The third four bytes describe the ending point of the arc. Arcs are drawn counter clockwise. The results are unpredictable if the three points do not really form an arc.
c	**circle**	Draws a circle whose center point is defined by the first four bytes, and whose radius is given as an integer in the following two bytes.
e	**erase**	Starts another frame of output.
f	**linemod**	Uses the following string, terminated by a new-line character, as the style for drawing further lines. The styles are dotted, solid, long-dashed, short-dashed, and dot-dashed.
l	**line**	Draws a line from the point designated by the next four bytes to the point designated by the following four bytes.

Instr	Sub	Description
m	**move**	The next four bytes designate a new current point.
n	**cont**	Draws a line from the current point to the point designated by the next four bytes.
p	**point**	Plots the point designated by the next four bytes.
s	**space**	The next four bytes designate the lower left corner of the plotting area; followed by four bytes for the upper right corner. The plot is magnified or reduced to fit the device as closely as possible.
t	**label**	Places the following ASCII string so that its first character falls on the current point. A new-line character terminates the string.

The space setting

```
space(0, 0, 480, 432);
```

exactly fills the plotting area with unity scaling for the IBM Personal Computer Graphics Printer. The upper limit is immediately outside the plotting area, which is taken to be square. Points outside the plotting area can be displayed on devices that do not have square displays.

Related Information

In this book: "plot" on page 3-296 and "TERM" on page 5-72.

The **graph** and **tplot** commands in *AIX Operating System Commands Reference.*

ports

Purpose

Describes the ports.

Description

The **ports** file contains the names and characteristics of all the system terminal ports. It provides a convenient means to associate values with named keyword parameters on a port-by-port basis, with defaults supplied as desired.

The **getty** process is the principal user of the information in this file. Since programs using this file look for specific keyword parameters and ignore all others, parameters other than those discussed here can be added to this file as necessary.

File Format

The **ports** file consists of one or more named stanzas usually separated by blank lines. Each stanza begins with its name followed by a colon, and contains assignments of values to keyword attributes. The values, in turn, may be alphanumeric strings or arbitrary character strings enclosed in double quotes.

Stanzas headed by the name **default** specify attribute-value pairs that are associated with all of the ports following it to the next **default** stanza. Explicit values within a port stanza override this association.

Port-Control Parameters

Most of the parameters in the **ports** file are port controls for login terminals. Because there are system defaults, specified in the **getty** process, it is not usually necessary to specify more than a few attributes in the **ports** file, as in the example. The port control parameters and their meanings are as follows:

enabled The **init** program uses this attribute to determine whether or not to create a logger on the port. If the port permits a logger, the value is *true*; otherwise the value is *false*. Note that **penable**, **pdisable**, and **phold** commands override the value specified. See the **penable** command in *AIX Operating System Commands Reference* for more information about these commands.

eof An octal integer specifying the character code that causes an end of file to be generated from the terminal. The system default is 004 (or 0x04), the ASCII EOT character, which is generated by **Ctrl-D**.

eol An optional and seldom used alternate line termination character to use in addition to the ASCII new-line (line-feed) character.

erase An octal integer specifying the character code that deletes the previously received character. The system default for the erase character is 010 (or 0x08), **Ctrl-h**, which is generated by the **Backspace** key on many terminals.

herald An arbitrary string, enclosed in double quotes, printed by the **getty** process to prompt for login. The C language \(backslash) escapes **\r**, **\n**, **\t**, **\b**, and **\f** are recognized as carriage return, new-line, tab, backspace, and formfeed, respectively.

imap This attribute is used by **getty** to set the terminal input map. If **imap** is not specified, **getty** resets the map to the system default.

intr An octal integer specifying the character code that interrupts the running process. The system default is 0177 (or 0x7f), which is usually generated by a key labeled Del or Rubout.

kill An octal integer specifying the character code that deletes the input line. system default for the kill character is 025 (or 0x15), **Ctrl**-u, which is the ASCII NAK character.

lock This attribute is used to request port locking. If the value is *true*, **init** creates a file in **/etc/locks** when the port is enabled and deletes the lock file when the port is disabled. Similarly, **penable** does not enable a port whose **lock** attribute is *true* when the corresponding lock file exists. Programs using the port for some other purpose (such as a link between processors) should check for an outstanding lock (and create a lock file, if necessary) before opening the port.

log This parameter causes logins to be recorded for a port on the console or in file **/usr/adm/sulog**. If **log=true**, all logins are reported, and if **log=root**, logins by **root** (superuser) are recorded. See **super** parameter on 4-119 for related information.

logger A character string giving the names the program to use at login. The default is **/bin/login**.

logmodes Console modes in effect while prompting for and reading in the user name. Modes are specified as a series of terminal options separated by a + (plus). Terminal options are as listed in the **stty** command. All listed modes not preceded with - (dash) are recognized. For example, the default **logmodes** parameter is specified as:

```
logmodes = cread+cs8+hupcl+echoe+echok
```

Because a speed value is not recognized in **logmodes** under any circumstances, the baud rate must be set with the **speed** parameter (see below).

min See the discussion of ICANON under "termio" on page 6-114.

omap This attribute is used by **getty** to set the terminal output map. If **omap** is not specified, **getty** resets the map to the system default.

owner Normally, when a port is logged in, the **login** program sets the logged-in user to be the owner of that port. Specifying an owner (either a UID or user name), the system manager forces the **getty** process to set ownership even before opening the port.

parity The values *odd*, *even*, and *none* cause the generation of odd, even, and no parity, respectively, while *inpck*, *ignpar*, and *parmrk* cause the checking input for parity errors, ignoring input characters with parity errors, and "marking" input parity errors as specified under "termio" on page 6-114. These values can be combined, as in `parity=odd+inpck`.

program If a value is specified, it is taken as the name of a program to run immediately after setting the logmodes. This feature is useful for establishing special purpose server ports that respond to a connection with a special protocol handler. If the special assignment **program=HOLD** is specified, no program runs on the port, but the logmodes, ownership, and protection are set and the port is held open. This is useful to keep the desired modes associated with a port that is occasionally seized for some special purpose.

protection Normally the protection on terminal is set to **rw--w--w-** (octal 622 or 0x192). The protection parameter overrides this default. The value can be set to an octal mask or a string such as **rw-rw-rw-** (octal 666 or 0x1b6).

quit An octal integer specifying the character code that causes the running process to abort. The system default is 026 (or 0x16), which is generated by pressing **Ctrl-V**.

runmodes Console modes in effect after the user name is read. The mode in which the port is left, specified similar to **logmodes**.

speed A decimal integer from the set {50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200} depending on the hardware capability.

super This parameter is passed on the logger in its environment. If **super=false**, then **login** does not allow **root** (the superuser) to log in on the port. This is useful for security on off-site terminal connections such as telephone links. (See **log** parameter, on page 4-118.)

term This parameter is passed to the logger and shell in their environment ("environment" on page 5-47) in the variable TERM. Some application software uses this information to determine the type of terminal the user is using.

time See the discussion of ICANON under "termio" on page 6-114.

timeout A decimal integer. If a user name is not specified before the given number of seconds, the **getty** process advances to the next port setting, or exits if all settings were exhausted.

Multiple values, separated by commas, can be specified as in the **speed**=*300,1200* line for dial-in terminals. This causes the port to be set up according to the first set of values for each attribute. If a framing error occurs, as a result of a user-generated BREAK on the line or a speed mismatch between the terminal and the set speed, the **getty** process advances to the next value on the list.

If multiple specifications occur for more than one parameter, all are advanced at the same time. Thus, a specification such as:

```
speed=300,1200
parity=none,odd+inpck
```

first tries the line at 300 baud with no parity. If a framing error occurs, it tries 1200 baud generation and checks for odd parity.

Other Port Parameters

The **ports** has all the port-specific information, not just information about loggers. The other parameters in the file are:

loc The location of the terminal connected to the port. This parameter is presently unused by any RT PC software. Because programs that access this file ignore keywords they do not use, helpful information can be added to keep all port-specific information together in one area.

printer The hard copy device used for output from optional word processing packages.

Example

The following example of a **ports** file illustrates some of its features:

```
default:
     enabled = false
     speed = 9600
     herald = "\033[H\033[J\rRT PC(noname)\r\nlogin: "
     printer = lp0
     term = dumb
     erase = 010
     kill = 025
     intr = 0177
```

```
/dev/console:
     loc = "console"
     term = hft
     enabled = true
     herald = "\033[H\033[J\rRT PC(/dev/console)\r\nlogin:"
```

Files

/etc/ports
/etc/locks

Related Information

In this book: "attributes" on page 4-20, "connect.con" on page 4-33, "environment" on page 5-47, and "termio" on page 6-114.

The **su, penable, getty, login, init**, and **stty** commands in *AIX Operating System Commands Reference.*

"Overview of International Character Support" in *IBM RT PC Managing the AIX Operating System.*

portstatus

Purpose

Centralizes commands to control ports.

Description

The **penable, pdisable,** and **phold** commands communicate with the **init** command (the process that controls loggers) through the **/etc/portstatus** file. See the **penable** command in *AIX Operating System Commands Reference* for more information on these commands. The format of this file is:

```
struct portstatus
   {  char ps_line[14];       /* device name */
           char ps_stat;      /* current status */
           char ps_rqst;      /* requested status */
};

#define ENABLE   01       /* spawn logger */
#define DISABLE  02       /* kill logger */
#define HOLD     04       /* spawn no new logger */
```

The fields are explained as follows:

ps_line Names the special file for the port, **tty01**, for example.

ps_rqst Used by the **penable** command to request changes in the enabling of a port.

ps_stat Used by the **init** command to show the current state of the port.

The **penable** command, with the **-i** flag specified, automatically initializes this file. If this file does not exist or is damaged, the **init** command cannot enable the ports properly.

File

/etc/portstatus

Related Information

The **penable** and **init** commands in *AIX Operating System Commands Reference*.

predefined

Purpose

Provides information for predefined devices.

Description

The **predefined** file contains information about hardware adapters and devices that is used by the **devices** command. Some of these devices may not be present in a particular configuration, but all of them are supported by the system. The **predefined** file contains information needed when adding one of these devices so that you do not have to supply the information yourself. The size of this file increases with new entries as additional licensed programs are installed in the system.

The **devices** command uses the information in this file to set up stanzas in the **system** and **qconfig** files when devices are added to the system. Note that information in this file has no effect on the system until it is moved to a stanza in the **system** or **qconfig** file.

The **predefined** file is similar in structure and content to the **system** file, and its stanzas can contain any of the keywords that are allowed in the **system** file. See "system" on page 4-139 for a description of the keywords that can appear in stanzas of these files.

The use of extended characters in the **predefined** file is not supported.

The **predefined** file contains several special stanzas:

defqueue Used by the **devices** command to create the queue stanza in the **qconfig** file when a printer or plotter is added.

defdevice Used by the **devices** command to create the device stanza in the **qconfig** file when a printer or plotter is added.

default Contains keywords and their values that are common to all device stanzas.

adpts Contains a list of adapters with the code number and description that the **devices** command uses to identify each one.

addrs Contains a list of adapters and their corresponding adapter type and address.

Example

The following shows sample entries of the **predefined** file.

```
defqueue:
        argname = none
        device = none

defdevice:
        file = /dev/none
        backend = /usr/lpd/piobe

default:
        modes = rw-rw-rw-
        owner = root

adpts:
        mp = 49
* IBM Mono Disp & Paral Prntr
        sp1 = 23
* IBM Ser/Par Adptr, Primary
        sp2 = 23
* IBM Ser/Par Adptr, Secondary
        rs232c1 = 35

addrs:
        2A03BC = mp
        2A0378 = sp1par
        2A0278 = sp2par
        2303F8 = sp1
        2302F8 = sp2
        351230 = rs232c1

5182:
* IBM PC Color Printer (5182)
        name = 5182
        nname = 5182
        driver = u5182
        crname = true
```

```
                minor = c
                vint = 4
                iodn =
                kaf_file = /etc/ddi/pprinter.kaf
                kaf_use = kparallel
                file = /etc/ddi/pprinter
                use = d5182
                noddi = false
                dtype = printer
        * Printer
                switchable = true
        * Coprocessor Device
                specproc = cfgaqcfg
                shared = false
                noduplicate = false
                dname = lp
                noshow = false
                aflag = true
        * adapter description
                adp = mp,spl,sp2
```

File

/etc/predefined

Related Information

In this book: "attributes" on page 4-20 and "system" on page 4-139.

profile

Purpose

Sets the user environment at login time.

Description

The **profile** file contains commands to be executed at login and variable assignments to be set and exported into the environment. The **/etc/profile** file contains commands executed by all users at login.

After the **login** program adds the LOGNAME (login name) and HOME (login directory) parameters to the environment, the commands in **$HOME/.profile** are executed, if it is present. The **.profile** file is the individual user profile that overrides the variables set in the **profile** file and is used to tailor the user environment variables set in **/etc/profile.** The **.profile** file is often used to set exported environment variables and terminal modes. The person who customizes the system can use **adduser** to set default **.profile** files in each user home directory. Users can tailor their environment as desired by modifying their **.profile** file.

Example

The following example is typical of a **/etc/profile** file:

```
# Set file creation mask
unmask 022
# Tell me when new mail arrives
MAIL=/usr/mail/$LOGNAME
# Add my /bin directory to the shell search sequence
PATH=/bin:/usr/bin:/etc::
# Set terminal type
TERM=hft
# Make some environment variables global
export MAIL PATH TERM
```

Files

$HOME/.profile
/etc/profile
/etc/profile.dos

Related Information

In this book: "environment" on page 5-47 and "TERM" on page 5-72.

The **env, login, mail, sh**, **stty**, and **su** commands in *AIX Operating System Commands Reference.*

qconfig

Purpose

Configures a printer queueing system.

Description

The **/etc/qconfig** file describes the queues and devices available for use by the **print** command, which places requests on a queue, and the **qdaemon** command, which removes requests from the queue and processes them. The **/etc/qconfig** file is an attribute file. See "attributes" on page 4-20 for details about the format of attribute files.

Some stanzas in this file describe queues, and other stanzas describe devices. Every queue stanza requires that one or more device stanzas immediately follow it in the file. The first queue stanza describes the default queue. The **print** command uses this queue when it receives no queue parameter.

The name of a queue stanza must be 1 to 3 characters long. The following table shows some of the field names along with some of the possible values that appear in this file:

acctfile Identifies the file used to save print accounting information. **FALSE**, the default, indicates suppress accounting. If the named file does not exist, no accounting is done.

argname Identifies the queue name identifier that is used in the **print** command to specify the queue.

device Identifies the symbolic name that refers to the device stanza.

discipline Defines the queue serving algorithm. **fcfs**, the default, means first come first served. **sjn** means shortest job next.

friend Indicates whether the backend updates the status file and responds to terminate signals. **TRUE** is the default. **FALSE** indicates it does not.

up Defines the state of the queue. **TRUE**, the default, indicates that it is running. **FALSE** indicates that it is not running.

If a field is omitted, its default value is assumed. The default values for a queue stanza are:

```
friend     = TRUE
discipline = fcfs
up         = TRUE
acctfile   = FALSE
```

Also, the default **argname** value is the name of the stanza preceded by a - (dash). The **device** field cannot be omitted.

The name of a device stanza is arbitrary. The fields that can appear in it stanza are:

access Specifies the type of access the backend has to the file specified by the **file** field. The value of **access** is **write** if the backend has write access to the file, or **both** if it has both read and write access. This field is ignored if the **file** field has the value **FALSE**.

align Specifies whether the backend sends a form-feed control before starting the job if the printer was idle. The default is **FALSE**.

backend Specifies the full path name of the backend, optionally followed by flags and parameters to be passed to it.

feed Specifies the number of separator pages to print when the device becomes idle, or the value **never**, which indicates that the backend is not to print separator pages.

file Identifies the special file where the output of backend is to be redirected. **FALSE**, the default, indicates no redirection. In this case, the backend opens the output file.

header Specifies whether a header page prints before each job or group of jobs. **never**, the default, indicates no header page at all. **always** means a header page before each job. **group** means a header before each group of jobs for the same user.

trailer Specifies whether a trailer page prints after each job or group of jobs. **never**, the default, means no trailer page at all. **always** means a trailer page after each job. **group** means a trailer page after each group of jobs for the same user.

The **qdaemon** places the information contained in the **feed, header, trailer**, and **align** fields into a status file that is sent to the backend. Backends that do not update the status file do not use the information it contains.

If a field is omitted, its default value is assumed. The **backend** field cannot be omitted. The default values in a device stanza are:

```
file    = FALSE
access  = write
feed    = never
header  = never
trailer = never
align   = FALSE
```

The **print** command automatically converts the ASCII **qconfig** file to binary when the binary version is missing or older than the ASCII version. The binary version is found in **/etc/qconfig.bin**.

Unlike the format of the **ports** file, the **qconfig** file format does not allow default stanzas.

Examples

1. The batch queue supplied with the AIX system might contain these stanzas:

```
bsh:
     argname = bsh
     friend = FALSE
     discipline = fcfs
     device = bshdev

bshdev:
     backend = /bin/sh
```

   To run a shell procedure called myproc using this batch queue, type:

```
print bsh myproc
```

   The queuing system runs the files one at a time, in the order submitted. The qdaemon process redirects standard input, standard output, and standard error to the **/dev/null** file.

2. To allow two batch jobs to run at once:

```
bsh:
     argname    = save
     friend     = FALSE
     discipline = fcfs
     device     = bsh1,bsh2

bsh1:
     backend    = /bin/sh

bsh2:
     backend    = /bin/sh
```

Files

/etc/qconfig
/etc/qconfig.bin
/usr/lpd/digest

Related Information

In this book: "attributes" on page 4-20.

The **print**, **lp**, and **qdaemon** commands in *AIX Operating System Commands Reference.*

rasconf

Purpose

Defines the reliability, availability, and serviceability (RAS) configuration file.

Description

The **rasconf** file defines attributes of the reliability, availability, and serviceability (RAS) system. Initially, RAS logging is inactive and must be activated before any RAS data can be collected.

This attribute file consists of stanzas that govern the actions of daemons associated with individual RAS devices. Each stanza name is the name of the associated RAS device.

The following attributes are valid:

file = *file* Specifies the file into which the daemon will write the RAS information.

size = *blocks* Specifies the maximum size, in 1024-byte blocks, to which the daemon will allow the file to grow.

Example

A typical **rasconf** file can contain the following:

```
/dev/osm:
  file = /usr/adm/ras/osm
  size = 100

/dev/error:
  file = /usr/adm/ras/errfile
  size = 50

/dev/trace:
  file = /usr/adm/ras/trcfile
  size = 80
  buffer = 6
```

File

/etc/rasconf

Related Information

In this book: "attributes" on page 4-20, "error" on page 6-15, "osm" on page 6-105, and "trace" on page 6-128.

The **errdemon** and **trace** commands in *AIX Operating System Commands Reference.*

sccsfile

Purpose

Contains the Source Code Control System (SCCS) information.

Description

The SCCS file is an ASCII file consisting of the following six logical parts:

checksum The sum value of all characters, except the characters in the first line.

delta table Information about each delta including type, SCCS identification (SID) date and time of creation, and comments.

user names Login names and numerical group IDs, or both, of users who are allowed to add or remove deltas from the SCCS file.

flags Definitions of internal keywords.

comments Descriptive information about the file.

body The actual text lines intermixed with control lines.

There are lines throughout an SCCS file that begin with the ASCII SOH (start of heading) character (octal 001). This character is called the **control character** and is represented graphically as @ (at sign) in the following text. Any line described in the following text not shown beginning with the control character cannot begin with the control character.

The **DDDDD** entries represent a 5-digit string (a number from 00000 to 99999).

The following describes each logical part of an **SCCS** file.

Checksum

The checksum is the first line of an SCCS file. The value of the checksum is the sum of all characters, except those of the first line. The **@h** designates a **magic number** of 064001 octal (or 0x6801). The format of the line is:

```
@hDDDDD
```

Delta Table

The delta table consists of a variable number of entries such as:

```
@sDDDDD/DDDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD . . .
@x DDDDD . . .
@g DDDDD . . .
@m <MR number>
.
.
.
@c <comments> . . .
.
.
.
@e
```

@s The first line which contains the number of lines inserted or deleted or unchanged respectively.

@d The second line which contains:

- The type of delta. **D** designates normal delta and **R** designates removed.
- The SCCS ID (SID) of the delta.
- The date and time the delta was created.
- The login name that corresponds to the real user ID at the time the delta was created.
- The serial numbers of the delta and its predecessor.

@i Contains the serial numbers of the deltas included. This line is optional.

@x Contains the serial numbers of deltas excluded. This line is optional.

@g Contains the serial numbers of the deltas ignored. This line is optional.

@m Optional lines, each one containing one modification request (MR) number associated with the delta.

@c Comment lines associated with the delta.

@e Ends the delta table entry.

User Names

The list of login names and numerical group IDs, or both, of users who can add deltas to the file, separated by new-line characters. The bracketing lines **@u** and **@U** surround the lines containing the list. An empty list allows any user to make a delta.

Flags

Flags are keywords used internally in the system. For more information about their use, see the **admin** command in *AIX Operating System Commands Reference*. The format of each flag line is:

@f <flag> <optional text>

The following flags are defined:

@ft <type of program>
@fv <program name>
@fi
@fb
@fm <module name>
@ff <floor>
@fc <ceiling>
@fd <default-sid>
@fn
@fj
@fl <lock-releases>
@fq <user defined>

The flags are used as follows:

b Allows the use of the *-b* option on the **get** command to cause a branch in the delta tree.

c Defines the highest release number, less than or equal to 9999, which can be retrieved by a **get** command for editing. This release number is called the ***ceiling*** release number.

d Defines the default SID to be used when one is not specified with a **get** command.

f Defines the lowest release number between 0 and 9999, which can be retrieved by a **get** command for editing. This release number is called the ***floor*** release number.

i Controls the error warning message "No ID keywords". When this flag is not present, this message is only a warning. When this flag is present, the file is not used and the delta is not made.

j Causes the **get** command to allow concurrent edits of the same base SID.

l Defines a list of releases that cannot be edited with **get** using the **-e** flag.

m Defines the first choice for the replacement text of the %M% identification keyword.

n Causes the **delta** command to insert a delta that applies no changes for those skipped releases when a delta for a new release is made. For example, delta 5.1 is made after delta 2.1, skipping releases 3 and 4. When this flag is omitted, it causes skipped releases to be completely empty.

q Defines the replacement for the %Q% identification keyword.

t Defines the replacement for the %Y% identification keyword.

v Controls prompting for **MR** numbers in addition to comments. If optional text is present, it defines an **MR** number validity checking program.

Comments

Typically, the comments section contains a description of the purpose of the file. Bracketing lines **@t** and **@T** surrounding text designate the comments section.

Body

The body section consists of control and text lines. Control lines begin with the control character, text lines do not. There are three kinds of control lines: **insert, delete,** and **end,** represented by:

@I DDDDD
@D DDDDD
@E DDDDD

respectively. The digit string is the serial number corresponding to the delta for the control line.

Related Information

The **admin, delta, get**, and **prs** commands in *AIX Operating System Commands Reference.*

system

Purpose

Identifies the system devices.

Description

The **system** file contains entries for currently configured real devices, virtual devices, and device managers. Some of this information is used by the Virtual Resource Manager (VRM) to establish the default running environment.

The **system** file is an attribute file containing stanzas that generally describe special files including information about AIX drivers or what non-standard VRM drivers are needed to support them. See "attributes" on page 4-20 for a description of attribute files. Also included is data for the **Define_Device** supervisor calls (SVCs) to the VRM if needed. See the **vrmconfig** program in *AIX Operating System Commands Reference* for more information.

Each special file named in the **system** file refers to a device driver entry in the **master** file. The driver entries specify the AIX device drivers to be configured. All drivers needed for specified special files are included, and those drivers marked as mandatory. Two driver entries may specify the same major device number where the same driver controls devices with different I/O code numbers.

The name of each stanza is the simple name of the special file.

The use of extended characters in the **system** file is not supported.

adp Indicates the valid adapter names for the device described in the stanza.

aflag Is a required keyword that indicates whether both an adapter and an AIX device driver are associated with the device described in the stanza. If the value is **false**, then the device has either an AIX device driver or a VRM device driver (and, in most cases, an adapter), but not both.

If **true**, then the **devices** command constructs the name of the **ddi** stanza when adding the device by concatenating the value of the **use** keyword, the adapter name that the user choses from the **adp** list, and a port number.

If **false**, then **devices** constructs the name of the ddi stanza by concatenating the value of the **use** keyword and a pseudo port number. Either a **maxminor** keyword or a **maxdev** keyword must be defined if the value of **aflag** is false. If the **maxminor** keyword is defined, which indicates that this device has only an AIX device driver, then the pseudo port number is the next unused integer between 0 and **maxminor - 1**. If the **maxdev** keyword is defined, which indicates that this device has only a VRM device driver, then the pseudo port number is the next unused integer between 0 and **maxdev - 1**.

The **showall** subcommand of the **devices** command displays the comment line that immediately follows the **aflag** definition as a description of the device or adapter.

crname Is a required keyword that indicates by a value of **true** or **false** whether the **devices** command should create a new driver name for the device.

ddi Specifies the hexadecimal bytes to be passed to the VRM **Define_Device** SVC. If a **customize helper** program is invoked, it determines the data to be passed. In that case, this attribute is not used.

dname Indicates the prefix name that is used to create the name of the device stanza in the **/etc/system** file and the special file in the **/dev** directory. The **devices** command uses this value when it creates a stanza name for a new special file.

driver Identifies the associated driver in the **master** file. This is mandatory in all device stanzas.

dtype Specifies the class of the device. Examples of this are **printer** and **disk**. The **devices** command displays this value when asking the user to choose a device class. It also uses this value to construct a list of device classes.

file Identifies the file that contains the stanzas included by the **use** attribute. This is the **/etc/ddi** file associated with the device.

iodn Specifies the I/O device number to use. If omitted, no **Define_Device** SVC is sent to the VRM.

kaf_file Indicates the name of the keyword **attribute** file to be used by the **customization helper** programs for the device described in the device stanza.

kaf_use Indicates the name of the stanza in the **kaf_file** that contains information about the attributes for the device.

maxdev Specifies the maximum number of IODNs supported by the VRM device driver for this adapter. This is equivalent to the maximum number of adapters that can be installed times the number of ports on each adapter.

minor Has a value of the form *cn*, where *c* is either **b** to denote a block device, or **c** to denote a character device. *n* is the minor device number.

modes Sets the protection bits for the special file, specified in the form *rwxrwxrwx*. Hyphens replace modes that are turned off, for example, `rw-r--r--`.

name Is a required keyword that identifies the type of the driver or the four-character name passed to the VRM using the **Define_Device** SVC. The default name is the first two and last two characters of the special file name.

native Identifies the model. A value **true** indicates IBM RT PC 6150, **false** indicates IBM RT PC 6151.

nname Is a required keyword that indicates the name of the device, adapter, and port number (if applicable).

nocopy The value **true** indicates that the associated VRM device driver stanza in **/etc/master** cannot contain the **copy** keyword, but must specify the **code** keyword instead.

noddi Indicates whether any device-dependent information is associated with the device. The value **true** indicates there is none. If **noddi = true**, then the **change** subcommand of the **devices** command does not allow the user to change device characteristics.

nodelete Indicates whether to delete the special file when this driver is removed. When this value is **true**, no attempt is made to delete the special file.

nodl Indicates whether the device can be deleted from the system by the **devices** command. The value **true** indicates the device cannot be deleted using this command.

noduplicate Indicates whether another device of this type can be added to the system. The value **true** indicates another device cannot be added.

noipl Indicates whether this stanza is processed at initial program load (IPL) time. When this value is **true**, this stanza is not processed at system initial program load (IPL) time.

noshow Indicates whether the **devices** command displays information from the stanza to the user. If **noshow = false**, then the **showdev** subcommand of the **devices** command displays all device characteristics and the **showall** subcommand displays the device.

nospecial When this value is **true**, no special file (**/dev** file) is to be created.

owner Specifies the name of the owner assigned to the **/dev** special file when it is created.

port Lists the number of ports on each adapter in the **adp** keyword. There is a one to one correspondence between each adapter and its number of ports. If the device being added is the adapter, **port** is the number of ports on the adapter. This keyword is required if the **aflag** keyword is true.

protocol Indicates whether this stanza is used by a protocol procedure. When this value is **true**, this stanza is used by a protocol procedure.

shared Sets the shared bit for the VRM **Define_Device** SVC. When this value is **true**, the shared bit is set.

specproc Indicates the name of the special processing routine that is to be invoked when customizing the system for the device. See "cfgadev" on page 3-15 for information about the application program interface to this feature.

switchable Indicates whether the device can be shared by the coprocessor. The value **true** indicates that it can be shared. If so, then the **devices** command displays this as a device that can be added to the coprocessor.

type Defines a device manager rather than a device driver when this value is **manager** and used with the **Define_Device** SVC.

uinfo Specifies the hexadecimal bytes to pass to the **CFUDRV** type **ioctl** call to configure a AIX device driver. If a customization helper program is invoked, it determines the data to pass. In that case, this attribute is not used.

use Identifies a stanza to be logically included in the current stanza. If a **file** attribute is present, the file is searched to find the indicated stanza for device dependent information. This keyword is required if the **file** keyword is present.

vdmgr Defines device drivers controlled by a manager. Values are the names of stanzas in the **system** file, separated by commas. The controlled drivers should include **nospecial=true**.

vint Identifies the virtual interrupt level to use. Level 4 is the only supported level. If not specified, the default value is **vint=4**.

Other parameters can be given for special **customization helper** programs.

Miscellaneous System Parameters

Both the **master** and the **system** files can have option lines in the **default** stanzas describing miscellaneous system customizing and tuning options. Options in the **system** file override those in the **master** file. See "master" on page 4-98 for a list of these parameters.

Other lines can be added as needed.

Example

The following is an excerpt of the **system** file entries.

```
* * system - actual devices

default:
        modes = rw-rw-rw-
        owner = root
        native = true

lp1:
* IBM PC Color Printer
        name = 5182
        crname = true
        minor = c1
        vint = 4
        iodn = 12003
        kaf_file = /etc/ddi/pprinter.kaf
        kaf_use = kparallel
        file = /etc/ddi/pprinter
        noddi = false
        dtype = printer
        nodelete = true
* Printer
        switchable = true
        specproc =cfgaqcfg
        shared = false
        noduplicate = false
        dname = lp
        noshow = false
        aflag = true
* IBM Mono Disp & Paral Prntr
        adp = mp,sp1,sp2
        use = d5182mp
        nname = 5182mp
        driver = u5182mp
```

```
lp2:
* IBM PC Color Printer  (5182)
        name = 5182
        crname = true
        minor = c2
        vint = 4
        iodn = 12004
        kaf_file = /etc/ddi/pprinter.kaf
        kaf_use = kparallel
        file = /etc/ddi/pprinter
        noddi = false
        dtype = printer
* Printer
        switchable = true
        specproc = cfgaqcfg
        shared = false
        noduplicate = false
        dname = lp
        noshow = false
        aflag = true
* IBM Ser/Par Adptr, Primary
        adp = mp,spq,sp2
        use = d5182spl
        nname = 5182spl
        driver = u5182spl

 lp3:
* IBM PC Color Printer (5182)
        name = 5182
        crname = true
        minor = c3
        vint = 4
        iodn = 12005
        kaf_file = /etc/ddi/pprinter.kaf
        kaf_use = /etc/ddi/pprinter
        noddi = false
        dtype = printer
* Printer
```

```
        switchable = true
        specproc = cfgaqcfg
        shared = false
        noduplicate = false
        dname = lp
        noshow = false
        aflag = true
* IBM Ser/Par Adptr, Secondary
        adp = mp,sp1,sp2
        use = d5182sp2
        nname = 5182sp2
        driver = u5182sp2
```

File

/etc/system

Related Information

In this book: "attributes" on page 4-20, "ddi" on page 4-43, "master" on page 4-98, and "predefined" on page 4-124.

The **config**, **devices**, and **vrmconfig** commands in *AIX Operating System Commands Reference.*

tar

Purpose

Describes the tape archive format.

Description

The **tar** command reads and writes tapes in tape archive format. A **tar** tape consists of several 512-byte logical blocks that can be grouped (on magnetic tape) into records, which are some constant multiple of 512-byte blocks long. Block in the following description means logical block.

The following is the format of a file header that precedes each disk file written on the tape:

```
struct {
     char name[100];
     char mode[8];
     char uid[8];
     char gid[8];
     char size[12];
     char mtime[12];
     char chksum[8];
     char linkflag;
     char linkname[100];
};
```

All fields, except **linkflag**, are ASCII null-terminated strings. Numeric fields can contain leading blanks. The fields have the following meanings:

chksum Contains a byte-by-byte sum of the entire header block assuming that the **chksum** field is all blanks.

gid Contains the group identification of the file, in octal.

linkflag Contains a 1 if this file is a link to a previous file on the the tape, otherwise null.

linkname Contains the name of a file if **linkflag** has a value of 1. The file named in this field is linked to the **name** file.

mode Contains the mode of the file, which includes the protection bits, setuid bits, setgid bits, and file type, in octal.

mtime Contains the modification time, in octal. This field gives the major/minor device number for special files.

name Contains the name of the file.

size Contains the size in bytes, in octal. This field is 0 for special files.

uid Contains the user identification of the file, in octal.

Unused bytes are null. Following the file header block are the data blocks of the file. The last block is null-padded if necessary. Two null blocks designate the end of the tape.

Directories and special files are treated in a slightly different way. A directory size is 0, meaning no data blocks follow, and its name ends with a / (slash). A special file is also written with 0 size. Its major/minor device number is in the **mtime** field.

Related Information

The **tar** command in *AIX Operating System Commands Reference.*

terminfo

Purpose

Describes terminals by capability.

Description

A **terminfo** file is a data base that describes terminals, defining their capabilities and their methods of operation. It is used by various programs, including the Extended Curses Library (**libcur.a**) and the **vi** editor. The information defined includes initialization sequences, padding requirements, cursor positioning, and other command sequences that control specific terminals.

This section explains the **terminfo** source file format. Before a **terminfo** source file can be used, it must be compiled using the **tic** command, which is described in *AIX Operating System Commands Reference.* You can edit and modify these source files, such as **/usr/lib/terminfo/ibm.ti**, which describes IBM terminals, and **/usr/lib/terminfo/dec.ti**, which describes DEC terminals.

See "TERM" on page 5-72 for a list of some of the terminals supported by predefined **terminfo** data base files and the corresponding values for the **TERM** environment variable.

Each **terminfo** entry consists of a number of fields separated by commas, ignoring any white space between commas. The first field for each terminal gives the various names the terminal is known separated by | (vertical bar) characters. The first name given should be the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names except the last should be in lowercase and not contain blanks. The last name can contain uppercase characters for readability.

Terminal names (except the last) should be chosen using the following conventions. A root name should be chosen to represent the particular hardware class of the terminal. This name should not contain hyphens, except to avoid synonyms that conflict with other names. Possible modes for the hardware or user preferences are indicated by appending a - (hyphen) and an indicator of the mode to the root name. Thus, a terminal in 132 column mode would be *term*-**w**. The following suffixes should be used where possible:

Suffix	Meaning	Example
-am	With automatic margins (usually default)	*term*-am
-c	Color mode	*term*-c
-w	Wide mode (more than 80 columns)	*term*-w
-nam	Without automatic margins	*term*-nam
-*n*	Number of lines on the screen	*term*-60
-na	No arrow keys (leave them in local)	*term*-na
-*n*p	Number of pages of memory	*term*-4p
-rv	Reverse video	*term*-rv

Types of Capabilities

Capabilities in **terminfo** are of three types: boolean, numeric, and string. Boolean capabilities indicate that the terminal has some particular feature. Boolean capabilities are true if the corresponding name is in the terminal description. Numeric capabilities give the size of the terminal or the size of particular delays. String capabilities give a sequence that can be used to perform particular terminal operations.

Entries can continue onto multiple lines by placing white space at the beginning of each subsequent line. Comments are included on lines beginning with the # (sharp sign) character.

List of Capabilities

The following table shows **VARIABLE**, which is the name the programmer uses to access the **terminfo** capability. The **CAP NAME** (capability name) is the short name used in the text of the data base, and is used by a person updating the database. The **I. CODE** is the 2-letter internal code used in the compiled data base, and always corresponds to a **termcap** capability name.

Capability names have no absolute length limit. An informal limit of five characters is adopted to keep them short and to allow the tabs in the source file **caps** to be aligned. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64 standard of 1979.

(P) Indicates that padding may be specified.

(G) Indicates that the string is passed through **tparm** with parameters as given (#*i*).

(*) Indicates that padding may be based on the number of lines affected.

(#*i*) Indicates the i^{th} parameter.

	CAP	I.	
VARIABLE	NAME	CODE	DESCRIPTION
Booleans:			
auto_left_margin	bw	bw	Indicates **cub1** wraps from column 0 to last column.
auto_right_margin	am	am	Indicates terminal has automatic margins.
beehive_glitch	xsb	xs	Indicates a terminal with f1=escape and f2=Ctrl-C.
ceol_standout_glitch	xhp	xs	Indicates standout not erased by overwriting.
eat_newline_glitch	xenl	xn	Ignores new-line character after 80 columns.
erase_overstrike	eo	eo	Erases overstrikes with a blank.
generic_type	gn	gn	Indicates generic line type (such as, dialup, switch)
hard_copy	hc	hc	Indicates hardcopy terminal.
has_meta_key	km	km	Indicates terminal has a meta key (shift, sets parity bit).
has_status_line	hs	hs	Indicates terminal has extra "status line."
insert_null_glitch	in	in	Indicates insert mode distinguishes nulls.
memory_above	da	da	Retains information above display in memory.
memory_below	db	db	Retains information below display in memory.
move_insert_mode	mir	mi	Indicates safe to move while in insert mode.
move_standout_mode	msgr	ms	Indicates safe to move in standout modes.
over_strike	os	os	Indicates terminal overstrikes.
status_line_esc_ok	eslok	es	Indicates escape can be used on the status line.
teleray_glitch	xt	xt	Indicates destructive tabs and blanks inserted while entering standout mode.
tilde_glitch	hz	hz	Indicates terminal cannot print ~ characters.
transparent_underline	ul	ul	Overstrikes with underline character.
xon_xoff	xon	xo	Indicates terminal uses xon/xoff handshaking.
Numbers:			
columns	cols	co	Specifies the number of columns in a line.
init_tabs	it	it	Provides tabs initially every # spaces.
lines	lines	li	Specifies the number of lines on screen or page
lines_of_memory	lm	lm	Specifies the number of lines of memory if > lines. A value of 0 indicates variable.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
magic_cookie_glitch	xmc	sg	Indicates number of blank characters left by **smso** or **rmso**.
padding_baud_rate	pb	pb	Indicates lowest baud where carriage return and line return padding is needed.
virtual_terminal	vt	vt	Indicates virtual terminal number.
width_status_lines	wsl	ws	Specifies the number of columns in status line.

Strings:

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
appl_defined_str	apstr	za	Application defined terminal string.
back_tab	cbt	bt	Back tab. (P)
bell	bel	bl	Produces an audible signal (bell). (P)
box_chars_1	box1	bx	Box characters primary set.
box_chars_2	box2	by	Box characters alternate set.
box_attr_1	batt1	Bx	Attributes for box_chars_1.
box_attr_2	batt2	By	Attributes for box_chars_2.
carriage_return	cr	cr	Indicates carriage return. (P*)
change_scroll_region	csr	cs	Changes scroll region to lines #1 through #2. (PG)
clear_all_tabs	tbc	ct	Clears all tab stops. (P)
clear_screen	clear	cl	Clears screen and puts cursor in home position. (P*)
clr_eol	el	ce	Clears to end of line. (P)
clr_eos	ed	cd	Clears to end of the display. (P*)
color_bg_0	colb0	d0	Background color 0 black.
color_bg_1	colb1	d1	Background color 1 red.
color_bg_2	colb2	d2	Background color 2 green.
color_bg_3	colb3	d3	Background color 3 brown.
color_bg_4	colb4	d4	Background color 4 blue.
color_bg_5	colb5	d5	Background color 5 magenta.
color_bg_6	colb6	d6	Background color 6 cyan.
color_bg_7	colb7	d7	Background color 7 white.
color_fg_0	colf0	c0	Foreground color 0 white.
color_fg_1	colf1	c1	Foreground color 1 red.
color_fg_2	colf2	c2	Foreground color 2 green.
color_fg_3	colf3	c3	Foreground color 3 brown.
color_fg_4	colf4	c4	Foreground color 4 blue.
color_fg_5	colf5	c5	Foreground color 5 magenta.

	CAP	I.	
VARIABLE	**NAME**	**CODE**	**DESCRIPTION**
color_fg_6	colf6	c6	Foreground color 6 cyan.
color_fg_7	colf7	c7	Foreground color 7 black.
column_address	hpa	ch	Sets cursor column. (PG)
command_character	cmdch	CC	Indicates terminal command prototype character can be set.
cursor_address	cup	cm	Indicates screen relative cursor motion row #1 col #2. (PG)
cursor_down	cud1	do	Moves cursor down one line.
cursor_home	home	ho	Moves cursor to home position (if no **cup**).
cursor_invisible	civis	vi	Makes cursor invisible.
cursor_left	cubl	le	Moves cursor left one space.
cursor_mem_address	mrcup	CM	Indicates memory relative cursor addressing.
cursor_normal	cnorm	ve	Makes cursor appear normal (undo **vs** or **vi**).
cursor_right	cufl	nd	Indicates nondestructive space (cursor right).
cursor_to_ll	ll	ll	Moves cursor to first column of last line (if no **cup**).
cursor_up	cuul	up	Moves cursor up one line (cursor up).
cursor_visible	cvvis	vs	Makes cursor very visible.
delete_character	dch1	dc	Deletes character. (P*)
delete_line	dl1	dl	Deletes line. (P*)
dis_status_line	dsl	ds	Disables status line.
down_half_line	hd	hd	Indicates subscript (forward 1/2 line feed).
enter_alt_charset_mode	smacs	as	Starts alternate character set. (P)
enter_blink_mode	blink	mb	Enables blinking.
enter_bold_mode	bold	md	Enables bold (extra bright) mode.
enter_ca_mode	smcup	ti	Begins programs that use **cup.**
enter_delete_mode	smdc	dm	Starts delete mode.
enter_dim_mode	dim	mh	Enables half-bright mode.
enter_insert_mode	smir	im	Starts insert mode.
enter_protected_mode	prot	mp	Enables protected mode.
enter_reverse_mode	rev	mr	Enables reverse video mode.
enter_secure_mode	invis	mk	Enables blank mode (characters invisible).
enter_standout_mode	smso	so	Begins standout mode.
enter_underline_mode	smul	us	Starts underscore mode.
erase_chars	ech	ec	Erases #1 characters. (PG)
exit_alt_charset_mode	rmacs	ae	Ends alternate character set. (P)
exit_attribute_mode	sgr0	me	Disables all attributes.
exit_ca_mode	rmcup	te	Ends programs that use **cup.**
exit_delete_mode	rmdc	ed	Ends delete mode.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
exit_insert_mode	rmir	ei	Ends insert mode.
exit_standout_mode	rmso	se	Ends stand out mode.
exit_underline_mode	rmul	ue	Ends underscore mode.
flash_screen	flash	vb	Indicates visible bell (may not move cursor).
font_0	font0	f0	Select font 0.
font_1	font1	f1	Select font 1.
font_2	font2	f2	Select font 2.
font_3	font3	f3	Select font 3.
font_4	font4	f4	Select font 4.
font_5	font5	f5	Select font 5.
font_6	font6	f6	Select font 6.
font_7	font7	f7	Select font 7.
form_feed	ff	ff	Ejects page (hardcopy terminal). (P*)
from_status_line	fsl	fs	Returns from status line.
init_1string	is1	i1	Initializes terminal.
init_2string	is2	i2	Initializes terminal.
init_3string	is3	i3	Initializes terminal.
init_file	if	if	Identifies file containing **is**.
insert_character	ich1	ic	Inserts character. (P)
insert_line	il1	al	Adds new blank line. (P*)
insert_padding	ip	ip	Inserts pad after character inserted. (P*)
key_backspace	kbs	kb	Sent by backspace key.
key_back_tab	kbtab	k0	Sent by backtab key.
key_catab	ktbc	ka	Sent by clear-all-tabs key.
key_clear	kclr	kC	Sent by clear-screen or erase key.
key_ctab	kctab	kt	Sent by clear-tab key.
key_command	kcmd	kc	Command request key.
key_command_pane	kcpn	kW	Command pane key.
key_dc	kdch1	kD	Sent by delete-character key.
key_dl	kdl1	kL	Sent by delete-line key.
key_do	kdo	ki	Do request key.
key_down	kcud1	kd	Sent by terminal down arrow key.
key_eic	krmir	kM	Sent by **rmir** or **smir** in insert mode.
key_end	kend	kw	End key.
key_eol	kel	kE	Sent by clear-to-end-of-line key.
key_eos	ked	kS	Sent by clear-to-end-of-screen key.
key_f0	kf0	k0	Sent by function key F0.
key_f1	kf1	k1	Sent by function key F1.
key_f2	kf2	k2	Sent by function key F2.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
key_f3	kf3	k3	Sent by function key F3.
key_f4	kf4	k4	Sent by function key F4.
key_f5	kf5	k5	Sent by function key F5.
key_f6	kf6	k6	Sent by function key F6.
key_f7	kf7	k7	Sent by function key F7.
key_f8	kf8	k8	Sent by function key F8.
key_f9	kf9	k9	Sent by function key F9.
key_f10	kf10	ka	Sent by function key F10.
key_f11	kf11	k<	Sent by function key F11.
key_f12	kf12	k>	Sent by function key F12.
key_help	khlp	kq	Help key.
key_home	khome	kh	Sent by home key.
key_ic	kich1	kI	Sent by insert character/enter insert mode key.
key_il	kil1	kA	Sent by insert line key.
key_left	kcub1	kl	Sent by terminal left arrow key.
key_ll	kll	kH	Sent by home-down key.
key_newline	knl	kn	New-line key.
key_next_pane	knpn	kv	Next-pane key.
key_npage	knp	kN	Sent by next-page key.
key_ppage	kpp	kP	Sent by previous-page key.
key_prev_cmd	kpcmd	kp	Sent by previous-command key.
key_quit	kquit	kQ	Quit key.
key_right	kcuf1	kr	Sent by terminal right arrow key.
key_scroll_left	kscl	kz	Scroll left.
key_scroll_right	kscr	kZ	Scroll right.
key_select	ksel	kU	Select key.
key_sf	kind	kF	Sent by scroll-forward/down key.
key_smap_in1	kmpf1	Kv	Input for special mapped key 1.
key_smap_out1	kmpt1	KV	Output for mapped key 1.
key_smap_in2	kmpf2	Kw	Input for special mapped key 2.
key_smap_out2	kmpt2	KW	Output for mapped key 2.
key_smap_in3	kmpf3	Kx	Input for special mapped key 3.
key_smap_out3	kmpt3	KX	Output for mapped key 3.
key_smap_in4	kmpf4	Ky	Input for special mapped key 4.
key_smap_out4	kmpt4	KY	Output for mapped key 4.
key_smap_in5	kmpf5	Kz	Input for special mapped key 5.
key_smap_out5	kmpt5	KZ	Output for mapped key 5.
key_sr	kri	kR	Sent by scroll-backward/up key.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
key_stab	khts	kT	Sent by set-tab key.
key_tab	ktab	ko	Tab key.
key_up	kcuu1	ku	Sent by terminal up arrow key.
keypad_local	rmkx	ke	Ends keypad transmit mode.
keypad_xmit	smkx	ks	Puts terminal in keypad transmit mode.
lab_f0	lf0	l0	Labels function key F0 if not F0.
lab_f1	lf1	l1	Labels function key F1 if not F1.
lab_f2	lf2	l2	Labels function key F2 if not F2.
lab_f3	lf3	l3	Labels function key F3 if not F3.
lab_f4	lf4	l4	Labels function key F4 if not F4.
lab_f5	lf5	l5	Labels function key F5 if not F5.
lab_f6	lf6	l6	Labels function key F6 if not F6.
lab_f7	lf7	l7	Labels function key F7 if not F7.
lab_f8	lf8	l8	Labels function key F8 if not F8.
lab_f9	lf9	l9	Labels function key F9 if not F9.
lab_f10	lf10	la	Labels function key F10 if not F10.
meta_on	smm	mm	Enables “meta mode” (8th bit).
meta_off	rmm	mo	Disables “meta mode.”
newline	nel	nw	Performs new-line function (behaves like CR followed by LF).
pad_char	pad	pc	Pads character (instead of NUL).
parm_dch	dch	DC	Deletes #1 characters. (PG*)
parm_delete_line	dl	DL	Deletes #1 lines. (PG*)
parm_down_cursor	cud	DO	Moves cursor down #1 lines. (PG*)
parm_ich	ich	IC	Inserts #1 blank characters. (PG*)
parm_index	indn	SF	Scrolls forward #1 lines. (PG)
parm_insert_line	il	AL	Adds #1 new blank lines. (PG*)
parm_left_cursor	cub	LE	Moves cursor left #1 spaces. (PG)
parm_right_cursor	cuf	RI	Moves cursor right #1 spaces. (PG*)
parm_rindex	rin	SR	Scrolls backward #1 lines. (PG)
parm_up_cursor	cuu	UP	Moves cursor up #1 lines. (PG*)
pkey_key	pfkey	pk	Programs function key #1 to type string #2.
pkey_local	pfloc	pl	Programs function key #1 to execute string #2.
pkey_xmit	pfx	px	Programs function key #1 to xmit string #2.
print_screen	mc0	ps	Prints contents of the screen.
prtr_off	mc4	pf	Disables the printer.
prtr_on	mc5	po	Enables the printer.
repeat_char	rep	rp	Repeats character #1 #2 times. (PG*)
reset_1string	rs1	r1	Resets terminal to known modes.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
reset_2string	rs2	r2	Resets terminal to known modes.
reset_3string	rs3	r3	Resets terminal to known modes.
reset_file	rf	rf	Identifies the file containing reset string.
restore_cursor	rc	rc	Restores cursor to position of last **sc**.
row_address	vpa	cv	Positions cursor to an absolute vertical position (set row). (PG)
save_cursor	sc	sc	Saves cursor position. (P)
scroll_forward	ind	sf	Scrolls text up. (P)
scroll_reverse	ri	sr	Scrolls text down. (P)
set_attributes	sgr	sa	Defines the video attributes. (PG9)
set_tab	hts	st	Sets a tab in all rows, current column.
set_window	wind	wi	Indicates current window is lines #1-#2 cols #3-#4.
tab	ht	ta	Tabs to next 8-space hardware tab stop.
to_status_line	tsl	ts	Moves to status line, column #1.
underline_char	uc	uc	Underscores one character and moves beyond it.
up_half_line	hu	hu	Indicates superscript (reverse 1/2 line-feed).
init_prog	iprog	iP	Locates the program for init.
key_a1	ka1	K1	Specifies upper left of keypad.
key_a3	ka3	K3	Specifies upper right of keypad.
key_b2	kb2	K2	Specifies center of keypad.
key_c1	kc1	K4	Specifies lower left of keypad.
key_c3	kc3	K5	Specifies lower right of keypad.
prtr_non	mc5p	pO	Enables the printer for #1 bytes.

Terminal capabilities have names. For instance, the fact that a terminal has automatic margins (such as, an automatic new-line when the end of a line is reached) is indicated by the capability **am**. Hence the description of the terminal includes **am**. Numeric capabilities are followed by the # (sharp sign) character and then the value. Thus the **cols#80** capability, which indicates the number of columns the terminal has, gives the value 80 for the terminal.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the 2-character code, an = (equal sign), and then a string ending at the following **,** (comma). A delay in milliseconds may appear anywhere in a string capability, enclosed between a **$< and a >** as in `el=\EK$<3>`, and padding characters are supplied by **tputs** to provide this delay. The delay can be either a number, such as 20, or a number followed by an * (asterisk), such as `3*`. An asterisk indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of **lines** affected. This is always 1, unless the terminal has **xenl** and the software uses it.) When an asterisk is specified, it is sometimes useful to give a delay of the form

a.b, such as, 3.5, to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there. Both **\E** and **\e** map to an **Escape** character, ^**x** maps to a **Ctrl-x** for any appropriate x, and the sequences **\n, \l, \r, \t, \b, \f, \s** give a new-line, line-feed, return, tab, backspace, form-feed, and space. Other escapes include \^ (backslash caret) for a ^ (caret), \ \ (backslash backslash) for a \ (backslash), **\,** (backslash comma) for a **,** (comma), **\:** (backslash colon) for a **:** (colon), and **\0** (backslash) for the null character. (**\0** will produce **\200**, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters can be given as 3 octal digits after a \ (backslash).

Sometimes, individual capabilities must be commented out. To do this, put a period before the capability name.

Preparing Descriptions

An effective way to prepare a terminal description is to imitate the description of a similar terminal in the **terminfo** file and add to the description gradually, using partial descriptions with **vi** to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of this file to describe it or bugs in **vi**. To test a new terminal description, set the environment variable TERMINFO to a path name of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/lib/terminfo**. A test to get the correct padding (if not known) is to edit the **/etc/passwd** file at 9600 baud, delete about 16 lines from the middle of the screen, then hit the **u** key several times quickly. If the terminal fails to display the result properly, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The following describe basic terminal capabilities:

am Indicates that the cursor moves to the beginning of the next line when it reaches the right margin. This capability also indicates whether the cursor can move beyond the bottom right corner of the screen.

bel Produces an audible signal (such as a bell or a beep).

bw Indicates that a backspace from the left edge of the terminal moves the cursor to the last column of the previous row.

clear Clears the screen leaving the cursor in the home position.

cols Specifies the number of columns on each line for the terminal.

cr Moves the cursor to the left edge of the current row. This code is usually carriage return (**Ctrl-M**).

cub1 Moves the cursor one space to the left, such as backspace.

cufl, cuul, and **cudl**
Moves the cursor to the right, up, and down, respectively.

hc Specifies a printing terminal. The **os** capability should also be specified.

lines Specifies the number of lines on a cathode ray tube (CRT) terminal.

os Indicates that when a character is displayed or printed in a position already occupied by another character, the terminal overstrikes the existing character, rather than replacing it with the new character. **os** applies to storage scope, printing, and APL terminals.

The **terminfo** initialization subroutine, **setupterm**, calls **termdef** to determine the number of lines and columns on the display. If **termdef** cannot supply this information, then **setupterm** uses the **lines** and **cols** values in the data base.

A point to note here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program should go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cufl** from the last column. The only local motion that is defined from the left edge is if **bw** is given, then a **cubl** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the **terminfo** file usually assumes that it is on by specifying **am**. If the terminal has a command that moves to the first column of the next line, that command can be given as **nel** (new-line). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf**, it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe printing terminals and simple CRT terminals. Thus, the Model 33 Teletype is described as:

```
33 | tty33 | tty | Model 33 Teletype,
    bel=^G, cols#72, cr=^M, cudl=^J, hc, ind=^J, os,
```

And another terminal is described as:

```
xxxx | x | xxxxxxxxx,
    am, bel=^G, clear=^Z, cols#80, cr=^M, cubl=^H, cudl=^J,
    ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with escapes similar to **printf %x** in it. For example, to address the cursor, the **cup** capability is given using two parameters: the row and column to address to. (Rows and columns are numbered starting with 0 and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameterized capabilties and their descriptions are:

cubl Backspaces the cursor one space.

cup Addresses the cursor using two parameters: the row and column to address. Rows and columns are numbered starting with 0 and refer to the physical screen visible to the user, not to memory.

cuul Moves the cursor up one line on the screen.

hpa and **vpa**
Indicates the cursor has row or column absolute cursor addressing, horizontal position absolute (**hpa**) and vertical position absolute (**vpa**).

Sometimes the **hpa** and **vpa** capabilities are shorter than the more general two parameter sequence and can be used in preference to **cup**. If there are parameterized local motions (such as, move **n** spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**.

ind*n* and **ri***n*
Scrolls text. These are parameterized versions of the basic capabilities **ind** and **ri**. *n* is the number of lines.

mrcup Indicates the terminal has memory-relative cursor addressing.

The parameter mechanism uses a stack and special **%** codes to manipulate it. Typically a sequence pushes one of the parameters onto the stack and then prints it in some format. Often more complex operations are necessary.

The % encodings have the following meanings:

%% Outputs a %. (percent sign).
%**d** Print pop() as in **printf** (numeric string from stack).
%**2d** Print pop() like %**2d** (minimum 2 digits output from stack).
%**3d** Print pop() like %**3d** (minimum 3 digits output from stack).
%**02d** Prints as in **printf** (2 digits output).
%**03d** Prints as in **printf** (3 digits output).
%**c** Print pop() gives %**c** (character output from stack).
%**s** Print pop() gives %**s** (string output from stack).

%**p**[*i*] Pushes the i^{th} parameter onto stack.
%**P**[*a-z*] Sets variable [*a-z*] to pop() (variable ouptput from stack).
%**g**[*a-z*] Gets variable [*a-z*] and pushes it onto the stack.
%'*c*' Character constant *c*.
%{*nn*} Integer constant *nn*.

%+ %- %* %/ %m
Arithmetic (%**m** is modulus): push(pop() *operation* pop())
%& %| %^ Bit operations: push(pop() *operation* pop())
%= %> %< Logical operations: push(pop() *operation* pop()).
%! %~ Unary operations push(*operation* pop())
%**i** Add 1 to first two parameters (for ANSI terminals).

%? *expr* %**t** *thenpart* %**e** *elsepart* %**;**
If-then-else. The %**e** *elsepart* is optional. You can make an else-if construct as with Algol 68:

%? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e b_4 %;

In this example, c_i denote conditions, and b_i denote bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get x - 5 one would use %gx%{5}%-.

Consider a terminal, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is **cup=6\E&a%p2%2dc%p1%2dY**.

Some terminals need the current row and column sent preceded by a ^**T** with the row and column simply encoded in binary, **cup=^T%p1%c%p2%c**. Terminals which use **%c** need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n**, ^**D**, and **\r**, as the system may change or discard them. (The library routines dealing with **terminfo** set terminal modes so that tabs are not expanded by the operating system; thus **\t** is safe to send.)

A final example is a terminal that uses row and column offset by a blank character, thus **cup=\E=%p1%' '%+%c%p2%' '%+%c**. After sending '**\E=**', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**. Similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing (0,0) to the top left corner of the screen, not of memory. (Thus, the **\EH** sequence on some terminals cannot be used for **home**.)

Area Clears

The following areas are used to clear large areas of the terminal:

ed Clears from the current position to the end of the display. This is defined only from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

el Clears from the current cursor postion to the end of the line without moving the cursor.

Insert/Delete Line

The following describes the insert and delete line capabilities:

csr Indicates the terminal has a scrolling region that can be set. This capability takes two parameters: the top and bottom lines of the scrolling region.

da Indicates the terminal can retain display memory above what is visible.

db Indicates the display memory can be retained below what is visible.

dl1 Indicates the line the cursor is on can be deleted. This done only from the first position on the line to be deleted. Additionally, the **dl** capability takes a single parameter indicating the number of lines to be deleted.

il1 Creates a new blank line before the line where the cursor is currently located and scrolls the rest of the screen down. This is done only from the first position of a line. The cursor then appears on the newly blank line. Additionally, the **il** capability can take a single parameter indicating the number of lines to insert.

rc Restores the cursor. When used after the **csr** capability, it gives an effect similar to delete line.

sc Saves the cursor. When used after the **csr** capability, it gives an effect similar to insert line.

wind Indicates the terminal has the ability to define a window as part of memory. This a parameterized string with 4 parameters: the starting and ending lines tn memory and the stating and ending columns in memory, in that order.

Insert/Delete Character

Generally, there are two kinds of intelligent terminals with respect to insert/delete character operations which can be described using the **terminfo** file. The most common insert/delete character operations affect only the characters on the current line and shift characters to the right and off the line. Other terminals make a distinction between typed and untyped blanks on the screen, shifting data displayed to insert or delete at a position on the screen occupied by an untyped blank, which is either eliminated or expanded to two untyped blanks. Clearing the screen and then typing text separated by cursor motions differentiates between the terminal types. You can determine the kind of terminal you have doing the following:

1. Type **abc**   **def** using local cursor movements, not spaces, between the **abc** and the **def**.
2. Position the cursor before the **abc** and place the terminal in insert mode. If typing characters causes the characters on the line to the right of the cursor to shift and exit the right side of the display, the terminal does not distinguish between blanks and untyped positions. If the **abc** moves to positions to the immediate left of the **def** and the characters move to the right on the line, around the end, and to the next line, the terminal is the second type. This is described by the **in** capability, which signifies insert null.

While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces) there are no known terminals whose insert mode cannot be described with the single attribute.

The **terminfo** file can describe both terminals having an insert mode and terminals that send a simple sequence to open a blank position on the current line. The following are used to describe insert or delete character capabilities:

dch1 Deletes a single character. **dch** with one parameter, **n** deletes **n** characters.

ech Erases **n** characters (equivalent to typing **n** blanks without moving the cursor) with one parameter.

ich1 Precedes the character to be inserted. Most terminals with an insert mode do not use this. Terminals that send a sequence to open a screen position should give it. (If the terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.)

ip Indicates post padding needed. This is given as a number of milliseconds. Any other sequence that may need to be sent after inserting a single character can be given in this capability.

mir Allows cursor motion while in insert mode. It is sometimes necessary to move the cursor while in insert mode to delete characters on the same line. Some terminals may not have this capability due to their handling of insert mode.

rmdc Exits delete mode.

rmir Ends insert mode.

smdc Enters delete mode.

smir Begins insert mode.

Note that if your terminal needs both to be placed into an insert mode and a special code to precede each inserted character, then both **smir/rmir** and **ichl** can be given, and both will be used. The **ich** capability, with one parameter, **n**, will repeat the effects of **ichl n** times.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes such as highlighting, underlining, and visible bells, these can be presented in a number of ways. Highlighting, such as ***standout mode***, presents a good, high contrast, easy-on-the-eyes format to add emphasis to error messages, and other attention getters. Underlining is another method to focus attention to a particular portion of the terminal. Visible bells include methods such as flashing the screen. The following capabilities describe highlighting, underlining, and visible bells for a terminal:

blink Indicates terminal has blink highlighting mode.

bold Indicates terminal has extra bright highlighting mode.

civis Causes the cursor to be invisible.

cnorm Causes the cursor to display normal. This capability reverses the effects of the **civis** and **cvvis** capabilities.

cvvis Causes the cursor to be more visible than normal when it is not on the bottom line.

dim Indicates the terminal has half-bright highlighting modes.

eo Indicates blanks erase overstrikes.

flash Indicates the terminal has a way of flashing the screen (a bell replacement) for errors without moving the cursor.

invis Indicates the terminal has blanking or invisible text highlighting modes.

msgr Indicates it is safe to move the cursor while in standout mode. Otherwise, programs using standout mode should exit standout mode before moving the cursor or sending a new-line. Some terminals automatically leave standout mode when they move to a new line or the cursor is addressed.

prot Indicates the terminal has protected highlighting mode.

rev Indicates the terminal has reverse video mode.

rmso Exits standout mode.

rmul Ends underlining.

sgr Sets attributes. **sgr0** turns off all attributes. Otherwise, if the terminal allows a sequence to set arbitrary combinations of modes, **sgr** takes 9 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 9 parameters are in this order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. (**sgr** can only support those modes for which separate attributes exist on a particular terminal.)

smcup and **rmcup**
Indicates the terminal needs to be in a special mode when running a program that uses any of the highlighting, underlining or visible bell capabilities. **smcup** enters this mode, while **rmcup** exits this mode. This need arises, for example, from terminals with more than one page of memory. If the terminal has only memory relative cursor addressing, and not screen relative cursor addressing, a screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used where **smcup** sets the command character to be used by the **terminfo** file.

smso Enters standout mode.

smul Begins underlining.

uc Underlines the current character and moves the cursor one space to the right.

ul Indicates the terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike.

xmc Indicates the number of blanks left if the capability to enter or exit standout mode leaves blank spaces on the screen.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local mode. If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcubl, kcufl, kcuul, kcudl,** and **khome**, respectively. If there are function keys such as **F0, F1, . . . , F10,** the codes they send can be given as **kf0, kf1, . . . , kf10.** If these keys have labels other than the default **F0** through **F10**, the labels can be given as **lf0, lf1, . . . , lf10.** The codes transmitted by certain other special keys can be given as:

kbs Indicates the **backspace** key.

kclr Indicates the **clear screen** or **erase** key.

kctab Indicates clear the tab stop in this column.

kdch1 Indicates the **delete character** key.

kdl1 Indicates the **delete line** key.

ked Indicates clear to end of screen.

kel Indicates clear to end of line.

khts Indicates set a tab stop in this column.

kich1 Indicates insert character or enter insert mode.

kil1 Indicates insert line.

kind Indicates scroll forward and/or down.

kll Indicates home down key (home is the lower left corner of the display, in this instance).

kmir Indicates exit insert mode.

knp Indicates next page.

kpp Indicates previous page.

ktbc Indicates the **clear all tabs** key.

ri Indicates scroll backward and/or up.

In addition, if the keypad has a 3-by-3 array of keys including the 4 arrow keys, the other 5 keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3-by-3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually **Ctrl-I**). A "backtab" command which moves left toward the previous tab stop can be given as **cbt**. By convention, if the terminal modes indicate that tabs are being expanded by the operating system rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every **n** spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the **tset** command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set.

Other capabilities include **is1**, **is2**, and **is3**, initialization strings for the terminal, **iprog**, the path name of a program to be run to initialize the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **terminfo** description. They are normally sent to the terminal, by the **tset** program, each time the user logs in. They are printed in the following order: **is1**, **is2**, setting tabs using **tbc** and **hts**; **if**; running the program **iprog**;

and finally **is3**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is2** and **if**. These strings are output by the **reset** program, which is used when the terminal starts behaving strangely, or not responding at all. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the terminal into 80-column mode would normally be part of **is2**, but it causes an annoying screen behavior and is not normally needed since the terminal is usually already in 80-column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Certain capabilities control padding in the terminal driver. These are primarily needed by hard copy terminals, and are used by the **tset** program to set terminal modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** cause the appropriate delay bits to be set in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Miscellaneous Strings

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used.

If the terminal has an extra “status line” that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally, the capability **hs** should be given. Special strings to go to the beginning of the status line and to return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The parameter **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to. If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string that turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, such as, **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form-feed), give this as **ff** (usually **Ctrl-L**).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, `tparm(repeat_char,'x',10)` is the same as `xxxxxxxxxx`.

If the terminal has a "meta key" which acts as a shift key, setting the eighth bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the eighth bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings that control an auxiliary printer connected to the terminal can be given in the following ways: **mc0** prints the contents of the screen, **mc4** turns off the printer, and **mc5** turns on the printer. When the printer is on, all text sent to the terminal is sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range can program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local mode; and **pfx** causes the string to be transmitted to the computer.

Indicating Terminal Problems

Terminals that do not allow ~ (tilde) characters to be displayed should indicate **hz**.

Terminals that ignore a line-feed character immediately after an **am** wrap should indicate **xenl**.

If **el** is required to get rid of standout (instead of merely writing normal text on top of it), **xhp** should be given.

Terminals for which tabs turn all characters moved to blanks should indicate **xt** (destructive tabs). This capability is interpreted to mean that it is not possible to position the cursor on top of the pads inserted for standout mode. Instead, it is necessary to erase standout mode using delete and insert line.

The terminal that is unable to correctly transmit the ESC (escape) or **Ctrl-C** characters has **xsb**, indicating that the **F1** key is used for ESC and **F2** for **Ctrl-C**.

Other specific terminal problems can be corrected by adding more capabilities of the form **x***x*.

Similar Terminals

If two terminals are very similar, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be cancelled by placing *xx@* to the left of the capability definition, where *xx* is the capability. For example, the entry:

```
term-nl, smkx@, rmkx@, use=term,
```

defines a terminal that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Data Base File Names

Compiled **terminfo** descriptions are placed in subdirectories under **/usr/lib/terminfo** in order to avoid performing linear searches through a single directory containing all of the **terminfo** description files. A given description file is stored in **/usr/lib/terminfo/***c***/***name*, where *name* is the name of the terminal, and *c* is the first letter of the terminal name. For example, the compiled description for the terminal `term4-nl` can be found in the file `/usr/lib/terminfo/t/term4-nl` You can create synonyms for the same terminal by making multiple links to the same compiled file. (See the **ln** command in *AIX Operating System Commands Reference* on how to create multiple links to a file.)

Example

The following entry, which describes a terminal, is among the entries in the **terminfo** file.

```
hft|High Function Terminal,
    cr=^M, cud1=\E[B, ind=\E[S, bel=^G, il1=\E[L, am, cub1=^H, ed=\E[J,
    el=\E[K, clear=\E[H\E[J, cup=\E[%ip1%d;%p2%dH, cols#80, lines=#25,
    dch1=\E[P, dl1=\E[M, home=\E[H,
    ich=\E[%p1%d@, ich1=\E[@, smir=\E[6, rmir=\E6,
    bold=\E[1m, rev=\E[7m, blink=\E[5m, invis=\E[8m,  sgr0=\E[0m,
    sgr=\E[%?%p1%t7;%;%?%p2%t4;%;%?%p3%t7;%;%?%p4%t5;%;%?%p6%t1;%;m,
    kcuu1=\E[A, kcud1=\E[B, kcub1=\E[D,
    kcuf1=\E[C, khome=\E[H, kbs=^H,
    cuf1=\E[C, ht=^I, cuu1=\E[A, xon,
    rmul=\E[m, smul=\E[4m, rmso=\E[m, smso=\E[7m,
    kpp=\E[150q, knp=\E[154q,
kf1=\E[001q, kf2=\E[002q, kf3=\E[003q, kf4=\E[004q,
kf5=\E[005q, kf6=\E[006q, kf7=\E[007q, kf8=\E[008q,
kf9=\E[009q, kf10=\E[010q,
bw,   eo,   it#8,   ms,
ch=\E%i%p1%dG,  ech=\E[%p15dx,
kdch1=\E[P,   kind=\E[151q,   kich1=\E[139q,   krmir \E[41,
kn=^M,   ko=^I,   ktab=\E[Z,   kri=\E[155q,
cub=\E[%p1%dD, cuf=\E[%p1%dC,  indn=\E[%p1dS, rin=\E[%p1%dT,
ri=\E[T,   cuu=\E[%p1%dA,
box1=\332\304\277\263\331\300\302\264\301\303\305,
box2=\311\315\273\272\274\310\313\271\312\314\316,
batt2=md,
colf0=\E[30m,   colf1=\E[31m,   colf2=\E[32m,  colf3=\E[33m,
colf4=\E[34m,   colf5=\E[35m,   colf6=\E[36m,  colf7=\E[37m,
colb0=\E[40m,   colb1=\E[41m,   colb2=\E[42m,  colb3=\E[43m,
colb4=\E[44m,   colb5=\E[45m,   colb6=\E[46m,  colb7=\E[47m,
```

Files

/usr/lib/terminfo/?/* Compiled terminal capability data base.

Related Information

In this book: "curses" on page 3-51, "Terminfo Level Subroutines" on page 3-57, "extended curses library" on page 3-131, "printf, fprintf, sprintf, NLprintf, NLfprintf, NLsprintf" on page 3-300,"termdef" on page 3-352 , and "TERM" on page 5-72.

The **display** and **tic** commands in *AIX Operating System Commands Reference.*

utmp, wtmp, .ilog

Purpose

Contains user and accounting information.

Synopsis

#include <utmp.h>

Description

When a user logs in successfully, the **login** program writes entries in **/etc/utmp**, the record of users logged into the system, and in **/usr/adm/wtmp** (if it exists), for use in accounting. On invalid login attemps (due to an incorrect login name or password), **login** makes entries in the **/etc/.ilog** file. When you log in as user **root** or **su** and the **/etc/.ilog** file is not empty, you see a message advising you to check the **/etc/.ilog** file for a record of unsuccessful login attempts.

The records in these files follow the **utmp** structure, which is defined in the **utmp.h** header file:

```
#define UTMP_FILE  "/etc/utmp"
#define WTMP_FILE  "/usr/adm/wtmp"
#define ILOG_FILE  "/etc/.ilog"

#define ut_name  ut_user
#define ut_id    ut_line

struct utmp  {
     char ut_user[8];         /* User login name */
     char ut_line[12];        /* device name (console, lnxx) */
     short  ut_pid;           /* process id */
     short  ut_type;          /* type of entry */
     struct exit_status {
        short e_termination;  /* Process termination status */
        short e_exit;         /* Process exit status */
        } ut_exit;            /* The exit status of a process */
                              /* marked as DEAD_PROCESS. */
        time_t  ut_time;      /* time entry was made */
};

/* Definitions for ut_type */

#define EMPTY          0
#define RUN_LVL        1
#define BOOT_TIME      2
#define OLD_TIME       3
#define NEW_TIME       4
#define INIT_PROCESS   5     /* Process spawned by "init" */
#define LOGIN_PROCESS  6     /* A "getty" process waiting for login */
#define USER_PROCESS   7     /* A user process */
#define DEAD_PROCESS   8
#define ACCOUNTING     9
#define UTMAXTYPE ACCOUNTING  /* Largest legal value of ut_type */

/* Special strings or formats used in the "ut_line" field when */
/* accounting for something other than a process. */
/* No string for the ut_line field can be more than 11 chars + */
/* a NULL in length. */

#define RUNLVL_MSG     "run-level ?"
#define BOOT_MSG       "system boot"
#define OTIME_MSG      "old time"
#define NTIME_MSG      "new time"
```

Files

/etc/utmp	Record of users logged into the system
/usr/adm/wtmp	Accounting information
/etc/.ilog	Record of invalid logins.

Related Information

The **login**, **who**, and **write** commands in *AIX Operating System Commands Reference.*

Chapter 5. Miscellaneous Facilities

About This Chapter

This chapter describes miscellaneous facilities, such as macro packages and character set tables.

ascii

Purpose

Maps the ASCII character set.

Synopsis

cat /usr/pub/ascii

Description

ASCII is a map of the ASCII character set that gives both the octal and hexadecimal equivalents for each character. This file can be printed as needed.

Note: This is neither the PC ASCII nor the RT ASCII character set. See "data stream" on page 5-5 for information about these character sets. The contents of this file are:

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel|
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb|
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us |
|040 sp |041 !  |042 "  |043 #  |044 $  |045 %  |046 &  |047 '  |
|050 (  |051 )  |052 *  |053 +  |054 ,  |055 -  |056 .  |057 /  |
|060 0  |061 1  |062 2  |063 3  |064 4  |065 5  |066 6  |067 7  |
|070 8  |071 9  |072 :  |073 ;  |074 <  |075 =  |076 >  |077 ?  |
|100 @  |101 A  |102 B  |103 C  |104 D  |105 E  |106 F  |107 G  |
|110 H  |111 I  |112 J  |113 K  |114 L  |115 M  |116 N  |117 O  |
|120 P  |121 Q  |122 R  |123 S  |124 T  |125 U  |126 V  |127 W  |
|130 X  |131 Y  |132 Z  |133 [  |134 \  |135 ]  |136 ^  |137 _  |
|140 `  |141 a  |142 b  |143 c  |144 d  |145 e  |146 f  |147 g  |
|150 h  |151 i  |152 j  |153 k  |154 l  |155 m  |156 n  |157 o  |
|160 p  |161 q  |162 r  |163 s  |164 t  |165 u  |166 v  |167 w  |
|170 x  |171 y  |172 z  |173 {  |174 |  |175 }  |176 ~  |177 del|
```

Figure 5-1. Octal ASCII Character Set

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0A nl	0B vt	0C np	0D cr	0E so	0F si
10 dle	11 dcl	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1A sub	1B esc	1C fs	1D gs	1E rs	1F us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C |	7D }	7E ~	7F del

Figure 5-2. Hexadecimal ASCII Character Set

File

/usr/pub/ascii

data stream

Purpose

Defines the data stream that an HFT virtual terminal uses in KSR mode.

Description

The IBM RT PC is capable of addressing 1024 distinct displayable characters. To designate these characters using 8-bit bytes, a code page convention is used. Each ***code page*** is an ordered set of up to 256 characters, which are called ***code points***. The first 32 code points of each code page are reserved for control codes and are the same for all code pages. The control codes do not have graphic representations, so each code page can have a maximum of 224 distinct graphic characters.

The remaining characters are divided into three code pages called ***P0***, ***P1***, and ***P2***. Two additional code pages called ***USER1*** and ***USER2*** are provided for user-defined symbols.

Code points in the range 32 to 127 (0x20 to 0x7F) of code page P0 represent the standard 7-bit US ASCII graphic symbols. P0 code points 128 to 255 (0x80 to 0xFF) and code points in pages P1 and P2 are collectively called ***extended characters***.

The following code page maps show the predefined graphic display symbols and their code point values within each of the three code pages.

First Hexadecimal Digit

Second Hexadecimal Digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	BLANK (SPACE)	0	@	P	‘	p	Ç	É	á	░	└	ð	Ó	-
1	SOH	DC1	!	1	A	Q	a	q	ü	æ	í	▒	┴	Ð	ß	±
2	STX	DC2	''	2	B	R	b	r	é	Æ	ó	▓	┬	Ê	Ô	=
3	ETX	DC3	#	3	C	S	c	s	â	ô	ú	│	├	Ë	Ò	¾
4	EOT	DC4	$	4	D	T	d	t	ä	ö	ñ	┤	─	È	õ	¶
5	ENQ	NAK	%	5	E	U	e	u	à	ò	Ñ	Á	┼	ı	Õ	§
6	ACK	SYN	&	6	F	V	f	v	å	û	ª	Â	ã	Í	µ	÷
7	BEL	ETB	'	7	G	W	g	w	ç	ù	º	À	Ã	Î	þ	¸
8	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿	©	╚	Ï	Þ	°
9	HT	EM	)	9	I	Y	i	y	ë	Ö	®	╣	╔	┘	Ú	¨
A	LF	SUB	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Û	•
B	VT	ESC	+	;	K	[	k	{	ï	ø	½	╗	╦	█	Ù	¹
C	FF	SS4	,	<	L	\	l	|	î	£	¼	╝	╠	▄	ý	³
D	CR	SS3	—	=	M	]	m	}	ì	Ø	¡	¢	═	¦	Ý	²
E	SO	SS2	.	>	N	∧	n	∼	Ä	×	«	¥	╬	Ì	¯	▮
F	S1	SS1	/	?	O	_	o	△	Å	ƒ	»	┐	¤	▀	´	BLANK 'FF'

Figure 5-3. Code Page P0

First Hexadecimal Digit

Second Hexadecimal Digit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	•	►	ã	ý	Ò	Ě	š	ť	ℓ	Ē	Į	ō	ų	“
1	SOH	DC1	☺	◄	β	þ	Ó	Č	Ľ	ŕ	ʼn	ǵ	ij	Ō	Ų	”
2	STX	DC2	☻	↕	Â	˛	Õ	Ć	Ň	ő	Ş	ĝ	Ĳ	œ	ŵ	=
3	ETX	DC3	♥	‼	À	¤	³	˘	Ř	ű	-	Ĝ	ĵ	Œ	Ŵ	-
4	EOT	DC4	♦	¶	Á	Đ	Û	Ę	Ś	Ť	ţ	ġ	ĵ	ŗ	ŷ	+
5	ENQ	NAK	♣	§	Ã	Ý	Ù	Ů	·	Ŕ	Ţ	Ġ	ķ	Ŗ	Ŷ	∞
6	ACK	SYN	♠	▬	ø	Þ	Ú	Ď	z̊	Ő	ā	Ģ	Ķ	ŝ	Ÿ	π
7	BEL	ETB	•	↨	Ê	®	ą	Ĺ	˛	Ű	Ā	ĥ	ĸ	Ŝ	©	Δ
8	BS	CAN	◘	↑	Ë	¾	ě	ľ	Ż	ă	ĉ	Ĥ	ļ	ŧ	¹	→
9	HT	EM	○	↓	È	¯	č	ň	ž	ğ	Ĉ	ħ	Ļ	Ŧ	TM	/
A	LF	SUB	◙	→	Í	¨	ć	đ	ź	İ	`	Ħ	I·	ũ	⅛	†
B	VT	ESC	♂	←	Î	´	ę	ř	Ž	Ă	ċ	ĩ	L·	Ũ	⅜	<
C	FF	SS4	♀	∟	Ï	=	ů	ś	Ź	Ğ	Ċ	Ĩ	ņ	ŭ	⅝	>
D	CR	SS3	♪	↔	Ì	õ	ď	˚	Ł	ˇ	ė	ī	Ņ	Ŭ	⅞	℞
E	SO	SS2	♫	▲	Ø	ı	ĺ	ł	Ń	˝	Ė	Ī	ŋ	ū	×	∈
F	SI	SS1	☼	▼	ð	Ô	Ą	ń	Š	ş	ē	į	Ŋ	Ū	’	∴

Figure 5-4. Code Page P1

First Hexadecimal Digit

Second Hexadecimal Digit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	↗	⊃	ω	$_1$	∂	^{6}	╨	τ	√					
1	SOH	DC1	↘	⊕	ν	$_2$	∿	^{7}	╤	Φ	╥					
2	STX	DC2	‡	└	o	$_3$	□	^{8}	╙	θ	n					
3	ETX	DC3	≠	⊗	ρ	$_4$	■	^{9}	╘	Ω						
4	EOT	DC4	∨	″	γ	$_5$	⍁	ϕ	╒	δ	¢					
5	ENQ	NAK	∧	=	ϑ	$_6$	Γ	Pts	╓	∞	⊔					
6	ACK	SYN	∥	Ψ	⎞	$_7$	L	┌	╫	φ	∨					
7	BEL	ETB	∠	ε	⎠	$_8$	Ξ	╡	╪	∈	↧					
8	BS	CAN	<	λ	≅	$_9$	∝	╢	▌	∩	∅					
9	HT	EM	>	η	ξ	⊥	△	╖	▐	≡						
A	LF	SUB	∓	ι	χ	◊̲	Y	╕	∝	≥						
B	VT	ESC	◇	⎛	υ	Ψ	≃	╛	β	≤						
C	FF	SS4	ʹ	⎝	ζ	Π	∼	╘	Γ	⌠						
D	CR	SS3	∫	‰	∫	Λ	^{0}	╞	π	⌡						
E	SO	SS2	∪	θ	ʅ	●	^{4}	╟	Σ	≈						
F	S1	SS1	⊂	κ	$_0$	ƀ	^{5}	╧	σ	•						

Figure 5-5. Code Page P2

Code Page Switching

Characters from code page P0 are represented in a character data stream by a single 8-bit byte corresponding to their code points.

Characters from other code pages are selected with single-shift controls. A ***single-shift control*** is one of the single-byte control codes SS1 (0x1F), SS2 (0x1E), SS3 (0x1D), and SS4 (0x1C). Each of these codes indicates that the following byte specifies a character from a code page other than P0. These control codes are called “single shifts” because they shift to another code page for a single character; that is, they are nonlocking shifts.

The byte that follows a single shift corresponds to the code point for the desired character, but with the most significant bit set. In other words, SS1, SS2, SS3, and SS4 must be followed by a byte in the range 0x80 to 0xFF. A single shift followed by 0x00 to 0x7F is not a valid code sequence. The single shift that is used specifies the upper or lower half of a code page as follows:

SS1 Lower half of code page P1 (P1 0x20 to 0x7F)
SS2 Upper half of code page P1 (P1 0x80 to 0xFF)
SS3 Lower half of code page P2 (P2 0x20 to 0x7F)
SS2 Upper half of code page P2 (P2 0x80 to 0xFF).

Note that in this scheme, code points in the range 0x00 to 0x7F (7-bit US ASCII) are unique in the data stream, and that they are never validly preceded by a single-shift control. This encoding scheme minimizes the changes necessary to existing software that is oriented toward 7-bit ASCII.

If a single-shift control is followed by a byte with the most significant bit set to zero (that is, a byte in the range 0x00 to 0x7F), then the single-shift prefix is ignored, and the byte is processed as an unprefixed character.

On both input and output, graphic character code points that are *not* prefixed with a single-shift control select a display symbol from the active graphic display set (G0 or G1) to be echoed or displayed on the screen. By default, both G0 and G1 are set to P0, and G0 is the active display set. The active graphic display set can be set to G0 or G1 with the SI and SO single-byte controls, respectively (see “Single-Byte Controls” on page 5-11). The mapping used for G0 and G1 can be set with the SG0 and SG1 control sequences (see “Multi-Byte Controls” on page 5-13).

On both input and output, valid graphic character code points that *are* prefixed with SS1, SS2, SS3, or SS4 bypass the active graphic display set and echo or display characters directly from code page P1 or P2.

Nonspacing Characters

For convenience when typing diacritical (accented) characters, a nonspacing or "dead" character facility is provided. A ***nonspacing character sequence*** is a two-key sequence consisting of one of the 13 diacritics followed by an alphabetic character or a space. The virtual terminal subsystem converts this two-key sequence into a single code point that may have a single-shift prefix. The resulting character is the alphabetic character with the specified diacritic mark. A diacritic followed by a space translates to the diacritic character itself.

The 13 valid diacritics are:

´	Acute Accent or Apostrophe	0xEF or 0x27
`	Grave Accent	0x60
^	Circumflex Accent	0x5E
¨	Umlaut Accent	0xF9
~	Tilde Accent	0x7E
ˇ	Caron Accent	0x1FF3
˘	Breve Accent	0x1E9D
˝	Double Acute Accent	0x1E9E
°	Overcircle Accent	0x1FFD
˙	Overdot Accent	0x1E85
¯	Macron Accent	0x1EA3
¸	Cedilla Accent	0xF7
˛	Ogonek Accent	0x1E87

If a nonspacing character and the following character do not combine to form a diacritical character in the set of predefined graphic symbols, then the diacritic is treated as a separate character code. For example, ~Q is treated as two characters, ~ and Q.

Note that nonspacing characters apply only to keyboard input and are not a feature of the data stream used by applications. Also, a diacritic must be explicitly designated as being nonspacing in the keyboard mapping for this facility to operate. None of the keys on the standard U.S. keyboard mapping are defined to be nonspacing characters. However, nonspacing characters can be defined. See "Set Keyboard Map (HFSKBD)" on page 6-36 for details.

Controls

Two types of controls are valid in a character stream data:

- Single-byte controls (also called ***control characters*** and ***control codes***), which have character values from 0 to 31 (0x00 to 0x1F)
- Multi-byte controls, which are also called escape sequences and control sequences.

Single-Byte Controls

The single-byte controls are common to all code pages. The following list shows the single-byte controls and their interpretation in KSR coded data. A line introducing each control gives its mnemonic, its code value, and its function.

- NUL, 0x00, (Null) has no terminal function.
- SOH, 0x01, (Start of Header) has no terminal function.
- STX, 0x02, (Start of Text) has no terminal function.
- ETX, 0x03, (End of Text) has no terminal function.
- EOT, 0x04, (End of Transmission) has no terminal function.
- ENQ, 0x05, (Enquiry) has no terminal function.
- ACK, 0x06, (Acknowledge) has no terminal function.
- BEL, 0x07, (Bell) causes an audible alarm to sound.
- BS, 0x08, (Backspace) moves the cursor position to the left one column, unless the cursor is at the left boundary of the presentation space. In that case, the cursor position does not change.
- HT, 0x09, (Horizontal Tab) moves the cursor position forward to the next tab stop. If the cursor is already in the last column of a line, then the cursor position does not change. Note that the CHT (cursor horizontal tab) multi-byte control performs a similar operation, but also performs line wrapping.
- LF, 0x0A, (Line Feed) if the LNM mode is reset, the line feed moves the cursor position down one line. If the LNM mode is set (default), the line feed is treated as a NEL and moves the cursor position to the first position of the next line. In either case, if the cursor is already on the last line of the PS, the PS lines scroll up one line. The top line of the PS disappears and a blank line is inserted as the new bottom line.
- VT, 0x0B, (Vertical Tab) moves the cursor position down to the next line that is defined as a vertical tab stop. Tabs stops are always set at the first and last lines of the PS. If the cursor was already on the last line of the PS and HFWRAP mode is not set, the cursor stays on the last line in the PS. If HFWRAP mode is set, the cursor moves to the top line in the PS. The column position does not change in any case.
- FF, 0x0C, (Form Feed) treated as a line end; see NEL.
- CR, 0x0D, (Carriage Return) if the CNM mode is reset (default), the carriage return moves the cursor position to the first character of the line indicated by the cursor. If the CNM mode is set, the carriage return is treated as an NEL and causes the cursor position to move to the first position of the next line. In this case, if the cursor is already on the last line of the PS, the PS lines scroll up one line. The top line of the PS disappears and a blank line is inserted as the new bottom line.

- SO, 0x0E, (Shift Out) maps the subsequently received graphic codes to display symbols according to the active G1 character set. See "display symbols" on page 5-24 for a list of the display symbols.
- SI, 0x0F, (Shift In) maps the subsequently received graphic codes to display symbols according to the active G0 character set. See "display symbols" on page 5-24 for a list of the display symbols.
- DLE, 0x10, (Data Link Escape) has no terminal function.
- DC1, 0x11, (Device Control 1) has no terminal function when output.
- DC2, 0x12, (Device Control 2) has no terminal function.
- DC3, 0x13, (Device Control 3) has no terminal function when output.
- DC4, 0x14, (Device Control 4) has no terminal function.
- NAK, 0x15, (Negative Acknowledgment) has no terminal function.
- SYN, 0x16, (Synchronous) has no terminal function.
- ETB, 0x17, (End of Block) has no terminal function.
- CAN, 0x18, (Cancel) has no terminal function.
- EM, 0x19, (End of Medium) has no terminal function.
- SUB, 0x1A, (Substitute) has no terminal function.
- ESC, 0x1B, (Escape) defines the beginning of a multi-byte control sequence as defined in "Multi-Byte Controls" on page 5-13.
- SS4, 0x1C, (Single Shift 4) causes the following byte is to be interpreted as belonging to the upper half of code page P2 (see "Code Page Switching" on page 5-9).
- SS3, 0x1D, (Single Shift 3) causes the following byte is to be interpreted as belonging to the lower half of code page P2.
- SS2, 0x1E, (Single Shift 2) causes the following byte is to be interpreted as belonging to the upper half of code page P1.
- SS1, 0x1F, (Single Shift 1) causes the following byte is to be interpreted as belonging to the lower half of code page P1.
- DEL, 0x7F, (Delete) has no terminal function.

Multi-Byte Controls

This section defines the code points and effects on the virtual terminal for multi-byte control sequences that are recognized in KSR mode. All of them begin with the ESC code (0x1B) followed by a [(0x5B) and include all subsequent bytes up to and including the first code in the range 0x40—0x7F. Any multi-byte control sequences not defined below are ignored. Invalid sequences return an error Device Status Report to the program. Multi-byte control sequences of more than 16 codes are considered invalid on receipt of the 17th code. The next code is not considered a part of that sequence. Also, numeric parameters in control sequences contain no more than 3 digits. The numeric value of the parameter may be incorrect if more than three digits are used, and the numeric value never exceeds 255.

Controls effect a virtual terminal's presentation space (PS) and its related cursor (pointer into the PS). The presentation space is a logical array of display symbols, N columns by M lines.

The following list gives the valid multi-byte control code sequences. A line introducing each control gives its mnemonic, its code sequence, and its function. The code sequence is shown in terms of ASCII characters. For example, the sequence **ESC A** represents two codes with a value of 0x1B41.

- CBT ESC [*PN* Z Cursor Back Tab

  Moves the cursor back the number of horizontal tab stops specified by *PN*. Tab stops are always set at the first and last columns of each line. If the cursor is already in the first column of a line and HFWRAP mode is set, the cursor moves to the last column. If AUTONL is also set, the cursor moves to the last column of the previous line. In this case, if the cursor is already on the first row of the PS, it moves to the last row.

- CHA ESC [*PN* G Cursor Horizontal Absolute

  Moves the cursor to the column specified by *PN*, unless the column exceeds the PS width. If the column exceeds the PS width, the cursor moves to the PS column farthest to the right.

- CHT ESC [*PN* I Cursor Horizontal Tab

  Moves the cursor position forward to the *PN*th following tab stop. If the cursor is already in the last column of a line and HFWRAP mode is set, then the cursor returns to the first column of the line. If AUTONL mode is also set, then the cursor moves to the first column of the next line. In this case, if the cursor is already on the last line of the PS, then the cursor moves to the first column of the first line. Note that the HT (horizontal tab) single-byte control does not cause wrapping to occur.

- CTC ESC [*PS* W Cursor Tab Stop Control

  0 Set a horizontal tab at cursor.
  1 Set a vertical tab at cursor.
  2 Clear a horizontal tab at cursor.
  3 Clear a vertical tab at cursor.

4 Clear all horizontal tabs on line.
5 Clear all horizontal tabs.
6 Clear all vertical tabs.

Sets or clears one or more tabulation stops according to the parameter specified. Tab stops on the first or last column cannot be cleared. When horizontal tab stops are set or cleared, the number of lines affected is all (if Tabulation Stop Mode is set) or one (if Tabulation Stop Mode is reset). This control does not change the position of characters already in the presentation space.

- CNL ESC [*PN* E Cursor Next Line

  Moves the cursor down the number of lines specified by *PN*, and over to the first position of that line. If the cursor was already on the bottom PS line and HFWRAP mode is not set, it is positioned at the beginning of that line. If HFWRAP mode is set, the cursor wraps from the bottom line to the top PS line.

- CPL ESC [*PN* F Cursor Preceding Line

  Moves the cursor back the number of lines specified by *PN*, and over to the first position of that line. If the cursor was already on the top PS line and HFWRAP mode is not set, the cursor is positioned at the beginning of that line. If HFWRAP mode is set, the cursor wraps from the top line to the bottom line of the PS.

- CPR ESC [*PN* ; *PN* R Cursor Position Report

  Reports the current cursor position. The first numeric parameter is the line number, and the second is the column. Line and column values are sent to the application as information. However, if the information is received by the virtual terminal, it is treated as a CUP control.

- CUB ESC [*PN* D Cursor Backward

  Moves the cursor backward on the line the specified number of columns. If this cursor movement exceeds the left PS boundary and HFWRAP mode is not set, the cursor stops at the leftmost PS position. If HFWRAP mode is set, the cursor wraps from the leftmost column to the rightmost column of the preceding PS line. In HFWRAP mode the cursor also wraps from the home to the rightmost bottom position of the PS.

- CUD ESC [*PN* B Cursor Down

  Moves the cursor down the number of lines specified by *PN.* If this cursor movement exceeds the bottom PS boundary and HFWRAP mode is not set, the cursor stops on the last PS line. If HFWRAP mode is set, the cursor wraps from the bottom line to the top line of the PS.

- CUF ESC [*PN* C Cursor Forward

  Moves the cursor forward on the line the specified number of columns. If this cursor movement exceeds the right PS boundary and HFWRAP mode is not set, the cursor stops at the rightmost PS position. If HFWRAP mode is set, the cursor wraps from the rightmost column to the leftmost column of the following line in the PS. In HFWRAP mode, the cursor also wraps from rightmost bottom position to the home position of the PS.

- CUP ESC [*PN* ; *PN* H Cursor Position

  Moves the cursor to the line specified by the first parameter, and to the column specified by the second parameter. If this movement crosses a PS boundary, the cursor stops at the PS boundary.

- CUU ESC [*PN* A Cursor up

  Moves the cursor up the specified number of lines. If this cursor movement exceeds the top PS boundary and HFWRAP mode is not set, the cursor stops on the first PS line. If HFWRAP mode is set, the cursor wraps from the top line to the bottom line in the PS.

- CVT ESC [*PN* Y Cursor Vertical Tab

  Moves the cursor down the number of vertical tab stops specified. Tab stops are assumed at the top and bottom PS lines. If there are not enough vertical tab stops in the PS and HFWRAP mode is not set, the cursor stops on the last line in the PS. If HFWRAP mode is set, the cursor wraps from the bottom line to the top line of the PS.

- DCH ESC [*PN* P Delete Character

  Deletes the cursor character and the following *PN*-1 characters on the line indicated by the cursor. The characters following the deleted characters on the line overlay the deleted character positions. The line is cleared from the end of the line to the edge of the presentation space. If the number of characters to be deleted exceeds the number of columns from the cursor to the PS right boundary, then all the characters from the cursor to the PS boundary are replaced with empty spaces and a DSR control sequence identifying an error is returned to the application.

- DL ESC [*PN* M Delete Line

  Deletes the line and the *PN*-1 following lines in the PS. The lines following the deleted lines are scrolled up *PN* lines and *PN* blanks lines are placed at the bottom of the PS. If there are less than *PN* lines from the line indicated by the cursor to the bottom of the PS, the line indicated by the cursor and all the following PS lines are replaced with empty lines.

- DSR ESC [*PN* n Device Status Report Request

6	Request Cursor Position Report
13	Error Report

  A request cursor position report (CPR) sends a cursor position report from the virtual terminal to the application. An error report is sent from the virtual terminal to the application when the virtual terminal receives an invalid control sequence. Error reports are private reports which conform to the ANSI standard for private parameters.

- DMI ESC ‘ (left quote) Disable Manual Input

  This control, when received in an output data stream, causes keyboard input to this terminal to be ignored. This control is ignored when received from the keyboard.

- EMI ESC b Enable Manual Input

  This control, when received in an output data stream, restarts keyboard input recognition and buffering if previously disabled with a DMI multi-byte control. This control is ignored when received from the keyboard.

EA	ESC [0 0	Erase to End of Area
	ESC [1 0	Erase from Start of Area
	ESC [2 0	Erase All of Area.

  This control is treated like an EL control sequence.

ED	ESC [0 J	Erase to End of Display
	ESC [1 J	Erase from Start of Display
	ESC [2 J	Erase All of Display.

  Erases certain characters within the PS. Erased characters are replaced with empty spaces. Erase to end of display erases the character indicated by the cursor and all following characters in the PS. Erase from start of display erases the first character of first line and the following characters up to and including the character indicated by the cursor. Erase all of display erases all the characters on the PS.

EF	ESC [0 N	Erase to End of Field
	ESC [1 N	Erase from Start of Field
	ESC [2 N	Erase All of Field.

  Erases certain characters between horizontal tab stops. Erased characters are replaced with empty spaces. Erase to end of field erases the character indicated by the cursor and all following characters before the next tab stop. Erase from start of field erases the character at the tab stop preceding the cursor an the following characters up to and including the character indicated by the cursor. Erase all of field erases the character at the tab stop preceding the cursor, and the following characters up to and including the character at the tab stop following the cursor. Tab stops are assumed at the first and last columns of the PS when executing this control.

EL	ESC [0 K	Erase to End of Line
	ESC [1 K	Erase from Start of Line
	ESC [2 K	Erase All of Line.

  Erases certain characters within a line. Erased characters are replaced with empty spaces. Erase to end of line erases the character indicated by the cursor and all following characters on the line. Erase from start of line erases the first character of first line and the following characters up to and including the character indicated by the cursor. Erase all of line erases all the characters on the line.

- ECH ESC [*PN* X Erase Character

  Erases the character indicated by the cursor and the following *PN*-1 characters on that line. Erased characters are replaced with empty spaces. If there are less than *PN* characters from the cursor to the PS right boundary, then the character indicated by the cursor and all the following characters on the line are replaced empty spaces.

- HTS ESC H Horizontal Tab Stop

  Sets a horizontal tab stop at the current horizontal position. If TSM is set, then the tab stop applies only to this line. If TSM is reset, then the tab stop applies to all PS lines. This control does not change the positioning of characters already in the presentation space.

- HVP ESC [*PN* ; *PN* f Horizontal and Vertical Position

  Moves the cursor to the line specified by the first parameter, and to the column specified by the second parameter. If this movement would cross a PS boundary, the cursor stops at the current PS boundary.

- ICH ESC [*PN* @ Insert Character

  Inserts *PN* empty spaces before the character indicated by the cursor. The string of characters starting with the character indicated by the cursor and ending with last character of the line are shifted *PN* columns to the right. Characters shifted past the PS right boundary are lost. The cursor does not move.

- IL ESC [*PN* L Insert Line

  Inserts *PN* empty lines before the line indicated by the cursor. The line indicated by the cursor is scrolled down. The cursor position on the screen is not affected.

- IND ESC D Index

  Moves cursor down one line. If the cursor was already on the bottom line of the PS, then the top line is lost, the other lines move up one line, and a blank line becomes the new bottom line.

- NEL ESC E Next Line

  Moves the cursor to the first position of the following line. If the cursor was already on the bottom line of the PS, then the top line is lost, the other lines move up one, and a blank line becomes the new bottom line.

- KSI ESC [*PS* p Keyboard Status Information

  The virtual terminal generates this control whenever **HFHOSTS** and **HFXLATKBD** are set and the status of the keyboard changes. Each selective parameter is the character-coded decimal value of a keyboard status byte. For example, if the keyboard has two status bytes, the control sequence is **ESC [** *xxx;yyy* **p**, where *xxx* is the value of the high-order byte and *yyy* is the value of the low-order byte. This is a private control that conforms to the ANSI standards for private control sequences. The virtual terminal display handler ignores this sequence whether it is received from the application or echoed. The values of the status bytes are described in "Untranslated Key Control" on page 6-56.

- PFK ESC [*PN* q PF Key Report

  The control sequence is sent by the virtual terminal to the application when a program function key (PFK) code is received from the keyboard. The parameter *PN* is a PF key number from 1 to 255. This is a private control that conforms to the ANSI standards for private control sequences. This sequence is ignored by the virtual terminal display handler whether received from the application or echoed.

- RCP ESC [u Restore Cursor Position

  Moves the cursor to the position saved by the last SCP control. If no SCP has been received, then the cursor position is set to the first character of the first line. This is a private control that conforms to the ANSI standards for private controls. This control has no terminal function when received from the keyboard.

- RI ESC L Reverse Index

  Moves the cursor up one line, unless the cursor is already on the PS top line. In that case, if HFWRAP mode is not set, then the cursor does not move. If HFWRAP mode is set, the cursor moves to the bottom line of the PS. The column position does not change.

- RIS ESC c Reset to Initial State

  Resets the virtual terminal to the state of a newly-opened virtual terminal: erases all PS data, places the cursor at the home position, resets graphic rendition to normal, resets subscripting and superscripting, and sets tab stops, modes, keyboard map, character maps and echo maps to their default values.

  Note: The RIS multi-byte control resets the VRM virtual terminal defaults, which are not necessarily the same as the defaults of an HFT device.

- RM ESC [*PS* l Reset Mode

20	LNM - Line Feed - New Line Mode
4	IRM - Insert Mode
12	SRM - Send Receive Mode (set ECHO off)
18	TSM - Tabulation Stop Mode
?21	CNM - Carriage Return - New Linė Mode
?7	AUTONL - Wrap character to following line when end of current line reached

Resets the modes specified in the parameter string. Multiple parameters must be separated by semicolons. The modes that can be reset are listed above with the appropriate parameter code. All other mode parameters are ignored.

TSM mode determines whether horizontal tabs apply identically to all line (TSM reset) or uniquely to each line on which they are set (TSM set).

- SCP ESC [s Save Cursor Position

  Saves the current cursor position. Any previously saved cursor position is lost. The cursor can be restored to this position with an RCP control. This is a private control that conforms to the ANSI standards for private controls. This control has no terminal function when received from the keyboard.

- SD ESC [*PN* T Scroll Down

  Moves all the PS lines down *PN* lines. The bottom *PN* lines are lost, and *PN* empty lines are put at the top of the presentation space. Physical cursor position does not change due to the scroll.

- SL ESC [*PN* SP @ Scroll Left

  Moves all the PS characters *PN* column positions to the left. The characters in the *PN* leftmost PS columns are lost, and empty spaces are put in the rightmost *PN* columns of all lines. Physical cursor position does not change due to the scroll.

- SR ESC [*PN* SP A Scroll Right

  Moves all the PS characters *PN* column positions to the right. The characters in the *PN* rightmost PS columns are lost, and empty spaces are put in the leftmost *PN* columns of all lines. Physical cursor position does not change due to the scroll.

- SU ESC [*PN* S Scroll Up

  Moves all the PS lines up *PN* lines. The top *PN* lines are lost, and *PN* empty lines are put at the bottom of the presentation space. The physical cursor position does not change due to the scroll.

- SGR ESC [*PS* m Set Graphic Rendition

0	Normal (none of attributes 1-9)
1	Bold or Bright
4	Underscore
5	Slow Blink
7	Negative (reverse image)
8	Cancelled On (invisible: set to background color)
10	Primary Font
11	First Alternate Font
12	Second Alternate Font
13	Third Alternate Font
14	Fourth Alternate Font
15	Fifth Alternate Font
16	Sixth Alternate Font
17	Seventh Alternate Font
30	Color palette entry 0 foreground

31	Color palette entry 1 foreground
32	Color palette entry 2 foreground
33	Color palette entry 3 foreground
34	Color palette entry 4 foreground
35	Color palette entry 5 foreground
36	Color palette entry 6 foreground
37	Color palette entry 7 foreground
40	Color palette entry 0 background
41	Color palette entry 1 background
42	Color palette entry 2 background
43	Color palette entry 3 background
44	Color palette entry 4 background
45	Color palette entry 5 background
46	Color palette entry 6 background
47	Color palette entry 7 background
90	Color palette entry 8 foreground
91	Color palette entry 9 foreground
92	Color palette entry 10 foreground
93	Color palette entry 11 foreground
94	Color palette entry 12 foreground
95	Color palette entry 13 foreground
96	Color palette entry 14 foreground
97	Color palette entry 15 foreground
100	Color palette entry 8 background
101	Color palette entry 9 background
102	Color palette entry 10 background
103	Color palette entry 11 background
104	Color palette entry 12 background
105	Color palette entry 13 background
106	Color palette entry 14 background
107	Color palette entry 15 background.

Causes the next characters received in the data stream or from the keyboard to have the display attributes specified by the parameter string. Any parameter not listed above is ignored.

The attributes corresponding to parameters 1 through 9 are cumulative. For example, specifying **underscore** and then specifying **blink** causes following characters to be underscored and blink. To reset one of these attributes, specify **normal** and then reinstate the desired parameters. Multiple parameters are processed in the order listed.

Whether the characters really have the requested attributes on the display depends on the capabilities of the physical display device used by the virtual terminal.

Note that switching between loaded fonts with the SGR sequence causes no data loss, but loading new fonts does cause data loss. (See "Untranslated Key Control" on page 6-56 for more information.)

Characters that cannot be displayed do not exist in the system.

- SG0A ESC (f Set G0 Character Set

  SG0B ESC , f Set G0 Character Set (Alternate form)

:	Unique One (User-defined)
;	Unique Two (User-defined)
<	P0 (Display Symbols 32-255)
=	P1 (Display Symbols 256-479)
>	P2 (Display Symbols 480-703)
?	User1 (Display Symbols 704-927)
@	User2 (Display Symbols 928-1023)

Designates the set of characters to use as the G0 set when the G0 set is invoked by SI. The default G0 set is the 224-character code page P0. Unique One and Unique Two may have unique definitions for each virtual terminal. When a virtual terminal is opened, these two sets are equivalent to <. See "Character Set Definition" on page 6-69 about defining Unique One and Unique Two.

- SG1A ESC) f Set G1 Character Set

  SG1B ESC - f Set G1 Character Set (Alternate)

:	Unique One (User-defined)
;	Unique Two (User-defined)
<	P0 (Display Symbols 32-255)
=	P1 (Display Symbols 256-479)
>	P2 (Display Symbols 480-703)
?	User1 (Display Symbols 704-927)
@	User2 (Display Symbols 928-1023)

Designates the set of characters to use as the G1 set when the G1 set is invoked by SO. The default G1 set is the 224-character code page P0. Unique One and Unique Two may have unique definitions for each virtual terminal. When a virtual terminal is opened, these two sets are equivalent to <. See "Character Set Definition" on page 6-69 about defining Unique One and Unique Two.

- SM ESC [*PS* h Set Mode

2 0	LNM - Line Feed - New Line Mode (default = 1)
4	IRM - Insert Replace Mode (default = 0)
1 2	SRM - Send Receive Mode (set echo off) (default = 0)
1 8	TSM - Tabulation Stop Mode (default = 0)
? 2 1	CNM - Carriage Return - New Line Mode (default = 0)
? 7	AUTONL - Wrap to next line when end of line reached (default = 1)

Sets the modes specified in the parameter string. Multiple parameters must be separated by semicolons. The modes that can be set are listed above with the appropriate parameter code. All other mode parameters are ignored.

SRM mode affects translated keyboard input handling. If SRM mode is set, translated keyboard input is never echoed by the virtual terminal, but is immediately returned to the application.

TSM mode determines whether horizontal tabs apply to all lines identically (TSM reset) or if horizontal tabs apply uniquely to each line on which they are set (TSM set).

- TBC ESC [*PS* g Tabulation Clear

  0 Horizontal tab at cursor column
  1 Vertical tab at line indicated by the cursor
  2 Horizontal tabs on line
  3 Horizontal tabs in presentation space
  4 Vertical tabs in presentation space.

  Clears tabulation stops specified by the parameters. Horizontal tab changes affect only the line indicated by the cursor if TSM is set, and horizontal tab changes affect all lines if TSM is reset. Any parameters not listed above are ignored. This control does not change the positioning of characters already in the presentation space.

- VTA ESC [r Virtual Terminal Addressability

  This private control sequence precedes a binary header and associated data that provide status information on the IBM 5081 Display Adapter.

- VTD ESC [x Virtual Terminal Data

  This private control sequence precedes a binary header and associated data. The block of data can be in formats other than character-coded data, such as binary format. See "Output" on page 6-61 for details about how this control sequence is used.

- VTL ESC [y Virtual Terminal Device Input

  This private control sequence precedes binary format input data from a mouse, tablet, LPFK, or valuator device. See "Input Device Report" on page 6-57 for details about how this control sequence is used.

- VTR ESC [w Virtual Terminal Raw Keyboard Input

  This private control sequence precedes "raw" (untranslated) keyboard input data, which is in a binary format. See "Untranslated Key Control" on page 6-56 for details about how this control sequence is used.

- VTS ESC I Vertical Tab Stop

  Sets a vertical tab stop at the line indicated by the cursor. This control does not change the positioning of characters already in the presentation space.

Related Information

In this book: “display symbols” on page 5-24 and “hft” on page 6-23.

Keyboard Description and Character Reference.

“Overview of International Character Support” in *IBM RT PC Managing the AIX Operating System.*

display symbols

Purpose

Defines the set of character symbols that can be displayed on an HFT display device in KSR mode.

Description

Each character code passed in KSR data is translated into one of 1024 10-bit display symbol codes. Codes 0 through 703 (0x2bf) are predefined to be common across all virtual terminals. Codes 704 (0x2c0) through 1023 (0x3ff) are reserved for user-defined extensions to the display symbol set. Display symbols 0 through 31 (0x1f) represent control functions and have no graphic representations.

Code pages P0, P1, and P2 contain all of the predefined characters. The first 32 code points of each are reserved for control characters and are common to all three code pages. The remaining characters are divided between P0, P1, and P2. Thus, each code page can have up to 224 distinct graphic characters.

In addition to the predefined code pages P0, P1, and P2, you can define two code pages called ***Unique One*** and ***Unique Two***. See "fonts" on page 4-68, "data stream" on page 5-5, and "Reconfigure (HFRCONF)" on page 6-31 for information you need to define such character sets.

The columns of the following tables represent:

Font Position
The position of the graphic display symbol within the font definition.

Code Page/Code Point
The code page of the symbol and the offset within that code page.

char String
The internal hexadecimal representation as a string of type **char**, including the single-shift control for characters in code pages other than P0.

NLchar Value
The value of the **NLchar** data type that corresponds to the character. The values 256—287 and 512—543 are not listed in this table because they correspond to control codes in code pages P1 and P2. See "NLchar" on page 3-276 for more information about this data type.

NCesc Esc Seq
The ASCII character or escape sequence that corresponds to the character after being translated by the **NCesc** macro. The **NLchar** values 256—287 and 512—543, which

correspond to control codes in code pages P1 and P2, translate to \< >, where two space characters (0x20) appear between the angle brackets. **NLchar** values outside the valid range translate to \<??>. See "conv" on page 3-39 and "NLescstr, NLunescstr, NLflatstr" on page 3-278 for related information.

The first table begins at font position 32 because the first 32 positions are reserved for the single-byte controls. The IBM PC ASCII graphic symbols for positions 1 through 31 are located at positions 257 through 287 and are not in any way associated with single-byte control functions.

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
32	Space		P0 32 (0x20)	0x20	32	Space
33	!	Exclamation Point	P0 33 (0x21)	0x21	33	!
34	"	Double Quote	P0 34 (0x22)	0x22	34	"
35	#	Number Sign	P0 35 (0x23)	0x23	35	#
36	$	Dollar Sign	P0 36 (0x24)	0x24	36	$
37	%	Percent Sign	P0 37 (0x25)	0x25	37	%
38	&	Ampersand	P0 38 (0x26)	0x26	38	&
39	′	Apostrophe, Acute Accent	P0 39 (0x27)	0x27	39	'
40	(	Left Parenthesis	P0 40 (0x28)	0x28	40	(
41	)	Right Parenthesis	P0 41 (0x29)	0x29	41	)
42	*	Asterisk	P0 42 (0x2a)	0x2a	42	*
43	+	Plus Sign	P0 43 (0x2b)	0x2b	43	+
44	,	Comma	P0 44 (0x2c)	0x2c	44	,
45	-	Hyphen, Minus Sign	P0 45 (0x2d)	0x2d	45	-
46	.	Period	P0 46 (0x2e)	0x2e	46	.
47	/	Slash	P0 47 (0x2f)	0x2f	47	/
48	0	Zero	P0 48 (0x30)	0x30	48	0
49	1	One	P0 49 (0x31)	0x31	49	1
50	2	Two	P0 50 (0x32)	0x32	50	2
51	3	Three	P0 51 (0x33)	0x33	51	3
52	4	Four	P0 52 (0x34)	0x34	52	4
53	5	Five	P0 53 (0x35)	0x35	53	5
54	6	Six	P0 54 (0x36)	0x36	54	6
55	7	Seven	P0 55 (0x37)	0x37	55	7
56	8	Eight	P0 56 (0x38)	0x38	56	8

Figure 5-6 (Part 1 of 8). Code Page P0

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
57	9	Nine	P0 57 (0x39)	0x39	57	9
58	:	Colon	P0 58 (0x3a)	0x3a	58	:
59	;	Semicolon	P0 59 (0x3b)	0x3b	59	;
60	<	Less Than Sign	P0 60 (0x3c)	0x3c	60	<
61	=	Equal Sign	P0 61 (0x3d)	0x3d	61	=
62	>	Greater Than Sign	P0 62 (0x3e)	0x3e	62	>
63	?	Question Mark	P0 63 (0x3f)	0x3f	63	?
64	@	At Sign	P0 64 (0x40)	0x40	64	@
65	A	a Uppercase	P0 65 (0x41)	0x41	65	A
66	B	b Uppercase	P0 66 (0x42)	0x42	66	B
67	C	c Uppercase	P0 67 (0x43)	0x43	67	C
68	D	d Uppercase	P0 68 (0x44)	0x44	68	D
69	E	e Uppercase	P0 69 (0x45)	0x45	69	E
70	F	f Uppercase	P0 70 (0x46)	0x46	70	F
71	G	g Uppercase	P0 71 (0x47)	0x47	71	G
72	H	h Uppercase	P0 72 (0x48)	0x48	72	H
73	I	i Uppercase	P0 73 (0x49)	0x49	73	I
74	J	j Uppercase	P0 74 (0x4a)	0x4a	74	J
75	K	k Uppercase	P0 75 (0x4b)	0x4b	75	K
76	L	l Uppercase	P0 76 (0x4c)	0x4c	76	L
77	M	m Uppercase	P0 77 (0x4d)	0x4d	77	M
78	N	n Uppercase	P0 78 (0x4e)	0x4e	78	N
79	O	o Uppercase	P0 79 (0x4f)	0x4f	79	O
80	P	p Uppercase	P0 80 (0x50)	0x50	80	P
81	Q	q Uppercase	P0 81 (0x51)	0x51	81	Q
82	R	r Uppercase	P0 82 (0x52)	0x52	82	R
83	S	s Uppercase	P0 83 (0x53)	0x53	83	S
84	T	t Uppercase	P0 84 (0x54)	0x54	84	T
85	U	u Uppercase	P0 85 (0x55)	0x55	85	U
86	V	v Uppercase	P0 86 (0x56)	0x56	86	V
87	W	w Uppercase	P0 87 (0x57)	0x57	87	W
88	X	x Uppercase	P0 88 (0x58)	0x58	88	X
89	Y	y Uppercase	P0 89 (0x59)	0x59	89	Y

Figure 5-6 (Part 2 of 8). Code Page P0

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
90	Z	z Uppercase	P0 90 (0x5a)	0x5a	90	Z
91	[	Left Bracket	P0 91 (0x5b)	0x5b	91	[
92	\	Reverse Slash	P0 92 (0x5c)	0x5c	92	\
93	]	Right Bracket	P0 93 (0x5d)	0x5d	93	]
94	^	Circumflex Accent, Up Arrow	P0 94 (0x5e)	0x5e	94	^
95	_	Underline, Low Line	P0 95 (0x5f)	0x5f	95	_
96	‘	Grave Accent, Left Single Quote	P0 96 (0x60)	0x60	96	‘
97	a	a Lowercase	P0 97 (0x61)	0x61	97	a
98	b	b Lowercase	P0 98 (0x62)	0x62	98	b
99	c	c Lowercase	P0 99 (0x63)	0x63	99	c
100	d	d Lowercase	P0 100 (0x64)	0x64	100	d
101	e	e Lowercase	P0 101 (0x65)	0x65	101	e
102	f	f Lowercase	P0 102 (0x66)	0x66	102	f
103	g	g Lowercase	P0 103 (0x67)	0x67	103	g
104	h	h Lowercase	P0 104 (0x68)	0x68	104	h
105	i	i Lowercase	P0 105 (0x69)	0x69	105	i
106	j	j Lowercase	P0 106 (0x6a)	0x6a	106	j
107	k	k Lowercase	P0 107 (0x6b)	0x6b	107	k
108	l	l Lowercase	P0 108 (0x6c)	0x6c	108	l
109	m	m Lowercase	P0 109 (0x6d)	0x6d	109	m
110	n	n Lowercase	P0 110 (0x6e)	0x6e	110	n
111	o	o Lowercase	P0 111 (0x6f)	0x6f	111	o
112	p	p Lowercase	P0 112 (0x70)	0x70	112	p
113	q	q Lowercase	P0 113 (0x71)	0x71	113	q
114	r	r Lowercase	P0 114 (0x72)	0x72	114	r
115	s	s Lowercase	P0 115 (0x73)	0x73	115	s
116	t	t Lowercase	P0 116 (0x74)	0x74	116	t
117	u	u Lowercase	P0 117 (0x75)	0x75	117	u
118	v	v Lowercase	P0 118 (0x76)	0x76	118	v
119	w	w Lowercase	P0 119 (0x77)	0x77	119	w
120	x	x Lowercase	P0 120 (0x78)	0x78	120	x
121	y	y Lowercase	P0 121 (0x79)	0x79	121	y
122	z	z Lowercase	P0 122 (0x7a)	0x7a	122	z

Figure 5-6 (Part 3 of 8). Code Page P0

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
123	{	Left Brace	P0 123 (0x7b)	0x7b	123	{
124	|	Logical OR	P0 124 (0x7c)	0x7c	124	|
125	}	Right Brace	P0 125 (0x7d)	0x7d	125	}
126	~	Tilde Accent	P0 126 (0x7e)	0x7e	126	~
127	Δ	Del	P0 127 (0x7f)	0x7f	127	Δ
128	Ç	c Cedilla Capital	P0 128 (0x80)	0x80	128	\<C,>
129	ü	u Umlaut Small	P0 129 (0x81)	0x81	129	\<u">
130	é	e Acute Small	P0 130 (0x82)	0x82	130	\<e'>
131	â	a Circumflex Small	P0 131 (0x83)	0x83	131	\<a^>
132	ä	a Umlaut Small	P0 132 (0x84)	0x84	132	\<a">
133	à	a Grave Small	P0 133 (0x85)	0x85	133	\<a`>
134	å	a Overcircle Small	P0 134 (0x86)	0x86	134	\<ao>
135	ç	c Cedilla Small	P0 135 (0x87)	0x87	135	\<c,>
136	ê	e Circumflex Small	P0 136 (0x88)	0x88	136	\<e^>
137	ë	e Umlaut Small	P0 137 (0x89)	0x89	137	\<e">
138	è	e Grave Small	P0 138 (0x8a)	0x8a	138	\<e`>
139	ï	i Umlaut Small	P0 139 (0x8b)	0x8b	139	\<i">
140	î	i Circumflex Small	P0 140 (0x8c)	0x8c	140	\<i^>
141	ì	i Grave Small	P0 141 (0x8d)	0x8d	141	\<i`>
142	Ä	a Umlaut Capital	P0 142 (0x8e)	0x8e	142	\<A">
143	Å	a Overcircle Capital	P0 143 (0x8f)	0x8f	143	\<Ao>
144	É	e Acute Capital	P0 144 (0x90)	0x90	144	\<E'>
145	æ	ae Diphthong Small	P0 145 (0x91)	0x91	145	\<ae>
146	Æ	ae Diphthong Capital	P0 146 (0x92)	0x92	146	\<AE>
147	o	o Circumflex Small	P0 147 (0x93)	0x93	147	\<o^>
148	ö	o Umlaut Small	P0 148 (0x94)	0x94	148	\<o">
149	ò	o Grave Small	P0 149 (0x95)	0x95	149	\<o`>
150	û	u Circumflex Small	P0 150 (0x96)	0x96	150	\<u^>
151	ù	u Grave Small	P0 151 (0x97)	0x97	151	\<u`>
152	ÿ	y Umlaut Small	P0 152 (0x98)	0x98	152	\<y">
153	Ö	o Umlaut Capital	P0 153 (0x99)	0x99	153	\<O">
154	Ü	u Umlaut Capital	P0 154 (0x9a)	0x9a	154	\<U">
155	ø	o Slash Small	P0 155 (0x9b)	0x9b	155	\<o/>

Figure 5-6 (Part 4 of 8). Code Page P0

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
156	£	English Pound Sign	P0 156 (0x9c)	0x9c	156	\<L=>
157	Ø	o Slash Capital	P0 157 (0x9d)	0x9d	157	\<0/>
158	×	Multiplication Sign	P0 158 (0x9e)	0x9e	158	\<x>
159	ƒ	Florin Sign	P0 159 (0x9f)	0x9f	159	\<f>
160	á	a Acute Small	P0 160 (0xa0)	0xa0	160	\<a'>
161	í	i Acute Small	P0 161 (0xa1)	0xa1	161	\<i'>
162	ó	o Acute Small	P0 162 (0xa2)	0xa2	162	\<o'>
163	ú	u Acute Small	P0 163 (0xa3)	0xa3	163	\<u'>
164	ñ	n Tilde Small	P0 164 (0xa4)	0xa4	164	\<n~>
165	Ñ	n Tilde Capital	P0 165 (0xa5)	0xa5	165	\<N~>
166	ª	Feminine Sign	P0 166 (0xa6)	0xa6	166	\<-a>
167	º	Masculine Sign	P0 167 (0xa7)	0xa7	167	\<-o>
168	¿	Inverted Question Mark	P0 168 (0xa8)	0xa8	168	\<?>
169	®	Registered Trademark	P0 169 (0xa9)	0xa9	169	\<r0>
170	¬	Logical Not	P0 170 (0xaa)	0xaa	170	\<-.>
171	½	One Half	P0 171 (0xab)	0xab	171	\<12>
172	¼	One Quarter	P0 172 (0xac)	0xac	172	\<14>
173	¡	Inverted Exclamation Sign	P0 173 (0xad)	0xad	173	\<!>
174	«	Left Angle Quotes	P0 174 (0xae)	0xae	174	\<{{>
175	»	Right Angle Quotes	P0 175 (0xaf)	0xaf	175	\<}}>
176	░	Quarter Hashed	P0 176 (0xb0)	0xb0	176	\<#1>
177	▒	Half Hashed	P0 177 (0xb1)	0xb1	177	\<#2>
178	▓	Full Hashed	P0 178 (0xb2)	0xb2	178	\<#3>
179	│	Vertical Bar	P0 179 (0xb3)	0xb3	179	\<S0>
180	┤	Right Side Middle	P0 180 (0xb4)	0xb4	180	\<S6>
181	Á	a Acute Capital	P0 181 (0xb5)	0xb5	181	\<A'>
182	Â	a Circumflex Capital	P0 182 (0xb6)	0xb6	182	\<A^>
183	À	a Grave Capital	P0 183 (0xb7)	0xb7	183	\<A`>
184	©	Copyright Symbol	P0 184 (0xb8)	0xb8	184	\<c0>
185	╣	Double Right Side Middle	P0 185 (0xb9)	0xb9	185	\<D6>
186	║	Double Vertical Bar	P0 186 (0xba)	0xba	186	\<D0>
187	╗	Double Upper Right Corner Box	P0 187 (0xbb)	0xbb	187	\<D9>
188	╝	Double Lower Right Corner Box	P0 188 (0xbc)	0xbc	188	\<D3>

Figure 5-6 (Part 5 of 8). Code Page P0

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
189	¢	Cent Sign	P0 189 (0xbd)	0xbd	189	\<c/>
190	¥	Yen Sign	P0 190 (0xbe)	0xbe	190	\<Y=>
191	┐	Upper Right Corner Box	P0 191 (0xbf)	0xbf	191	\<S9>
192	└	Lower Left Corner Box	P0 192 (0xc0)	0xc0	192	\<S1>
193	┴	Bottom Side Middle	P0 193 (0xc1)	0xc1	193	\<S2>
194	┬	Top Side Middle	P0 194 (0xc2)	0xc2	194	\<S8>
195	├	Left Side Middle	P0 195 (0xc3)	0xc3	195	\<S4>
196	─	Center Box Bar	P0 196 (0xc4)	0xc4	196	\<S.>
197	┼	Intersection	P0 197 (0xc5)	0xc5	197	\<S5>
198	ã	a Tilde Small	P0 198 (0xc6)	0xc6	198	\<a~>
199	Ã	a Tilde Capital	P0 199 (0xc7)	0xc7	199	\<A~>
200	╚	Double Lower Left Corner Box	P0 200 (0xc8)	0xc8	200	\<D1>
201	╔	Double Upper Left Corner Box	P0 201 (0xc9)	0xc9	201	\<D7>
202	╩	Double Bottom Side Middle	P0 202 (0xca)	0xca	202	\<D2>
203	╦	Double Top Side Middle	P0 203 (0xcb)	0xcb	203	\<D8>
204	╠	Double Left Side Middle	P0 204 (0xcc)	0xcc	204	\<D4>
205	═	Double Center Box Bar	P0 205 (0xcd)	0xcd	205	\<D.>
206	╬	Double Intersection	P0 206 (0xce)	0xce	206	\<D5>
207	¤	International Currency Symbol	P0 207 (0xcf)	0xcf	207	\<o*>
208	ð	eth Icelandic Small	P0 208 (0xd0)	0xd0	208	\<d+>
209	Ð	eth Icelandic Capital	P0 209 (0xd1)	0xd1	209	\<D+>
210	Ê	e Circumflex Capital	P0 210 (0xd2)	0xd2	210	\<E^>
211	Ë	e Umlaut Capital	P0 211 (0xd3)	0xd3	211	\<E">
212	È	e Grave Capital	P0 212 (0xd4)	0xd4	212	\<E`>
213	ı	Small i Dotless	P0 213 (0xd5)	0xd5	213	\<i>
214	Í	i Acute Capital	P0 214 (0xd6)	0xd6	214	\<I'>
215	Î	i Circumflex Capital	P0 215 (0xd7)	0xd7	215	\<I^>
216	Ï	i Umlaut Capital	P0 216 (0xd8)	0xd8	216	\<I">
217	┘	Lower Right Corner Box	P0 217 (0xd9)	0xd9	217	\<S3>
218	┌	Upper Left Corner Box	P0 218 (0xda)	0xda	218	\<S7>
219	█	Bright Character Cell	P0 219 (0xdb)	0xdb	219	\<B>
220	▄	Bright Character Cell - Lower Half	P0 220 (0xdc)	0xdc	220	\<B2>
221	¦	Broken Vertical Bar	P0 221 (0xdd)	0xdd	221	\<BO>

Figure 5-6 (Part 6 of 8). Code Page P0

Font			Code Page	char	NLchar	NCesc
Position	Character		Code Point	String	Value	Esc Seq
222	Ì	i Grave Capital	P0 222 (0xde)	0xde	222	\<I`>
223	▀	Bright Character Cell - Upper Half	P0 223 (0xdf)	0xdf	223	\<B8>
224	Ó	o Acute Capital	P0 224 (0xe0)	0xe0	224	\<O'>
225	β	s Sharp Small	P0 225 (0xe1)	0xe1	225	\<ss>
226	Ô	o Circumflex Capital	P0 226 (0xe2)	0xe2	226	\<O^>
227	Ò	o grave capital	P0 227 (0xe3)	0xe3	227	\<O`>
228	õ	o Tilde Small	P0 228 (0xe4)	0xe4	228	\<o~>
229	Õ	o Tilde Capital	P0 229 (0xe5)	0xe5	229	\<O~>
230	µ	Mu Small, Micro Symbol	P0 230 (0xe6)	0xe6	230	\<&m>
231	þ	Thorn Icelandic Small	P0 231 (0xe7)	0xe7	231	\<Ip>
232	Þ	Thorn Icelandic Capital	P0 232 (0xe8)	0xe8	232	\<IP>
233	Ú	u Acute Capital	P0 233 (0xe9)	0xe9	233	\<U'>
234	Û	u Circumflex Capital	P0 234 (0xea)	0xea	234	\<U^>
235	Ù	u Grave Capital	P0 235 (0xeb)	0xeb	235	\<U`>
236	ý	y Acute Small	P0 236 (0xec)	0xec	236	\<y'>
237	Ý	y Acute Capital	P0 237 (0xed)	0xed	237	\<Y'>
238	¯	Overbar	P0 238 (0xee)	0xee	238	\<-->
239	´	Acute Accent	P0 239 (0xef)	0xef	239	\<_'>
240	−	Syllable Hyphen	P0 240 (0xf0)	0xf0	240	\<^->
241	±	Plus Or Minus Sign	P0 241 (0xf1)	0xf1	241	\<+->
242	‗	Double Underscore	P0 242 (0xf2)	0xf2	242	\<__>
243	¾	Three Fourths	P0 243 (0xf3)	0xf3	243	\<34>
244	¶	Paragraph Symbol	P0 244 (0xf4)	0xf4	244	\<|P>
245	§	Section Symbol	P0 245 (0xf5)	0xf5	245	\<|S>
246	÷	Division Sign	P0 246 (0xf6)	0xf6	246	\<:->
247	¸	Cedilla Accent	P0 247 (0xf7)	0xf7	247	\<_,>
248	°	Degree Symbol, Overcircle Accent	P0 248 (0xf8)	0xf8	248	\<o>
249	¨	Umlaut Accent	P0 249 (0xf9)	0xf9	249	\<_">
250	·	Middle Dot, Product Dot	P0 250 (0xfa)	0xfa	250	\<..>
251	¹	Superscript 1	P0 251 (0xfb)	0xfb	251	\<^1>
252	³	Superscript 3	P0 252 (0xfc)	0xfc	252	\<^3>

Figure 5-6 (Part 7 of 8). Code Page P0

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
253	^{2}	Superscript 2	P0 253 (0xfd)	0xfd	253	\<^2>
254	▌	Vertical Solid Rectangle	P0 254 (0xfe)	0xfe	254	\<[]>
255		Required Space	P0 255 (0xff)	0xff	255	\<##>

Figure 5-6 (Part 8 of 8). Code Page P0

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
256	· Spanish Middle Dot	P1 32 (0x20)	0x1fa0	288	\<_.>
257	☺ Smiling Face	P1 33 (0x21)	0x1fa1	289	\<:)>
258	☻ Dark Smiling Face	P1 34 (0x22)	0x1fa2	290	\<(:>
259	♥ Heart	P1 35 (0x23)	0x1fa3	291	\<SH>
260	♦ Diamond	P1 36 (0x24)	0x1fa4	292	\<SD>
261	♣ Club	P1 37 (0x25)	0x1fa5	293	\<SC>
262	♠ Spade	P1 38 (0x26)	0x1fa6	294	\<SS>
263	• Bullet	P1 39 (0x27)	0x1fa7	295	\<@>
264	◘ Reverse Video Bullet	P1 40 (0x28)	0x1fa8	296	\<@#>
265	○ Circle	P1 41 (0x29)	0x1fa9	297	\<O>
266	◙ Reverse Video Circle	P1 42 (0x2a)	0x1faa	298	\<O#>
267	♂ Male Symbol	P1 43 (0x2b)	0x1fab	299	\<o%>
268	♀ Female Symbol	P1 44 (0x2c)	0x1fac	300	\<o+>
269	♪ Eighth Note	P1 45 (0x2d)	0x1fad	301	\<d~>
270	♫ Sixteenth Note	P1 46 (0x2e)	0x1fae	302	\<d=>
271	☼ Sun	P1 47 (0x2f)	0x1faf	303	\<*>
272	► Right Solid Triangle	P1 48 (0x30)	0x1fb0	304	\<#}>
273	◄ Left Solid Triangle	P1 49 (0x31)	0x1fb1	305	\<{#>
274	↕ Bidirectional Vertical Arrow	P1 50 (0x32)	0x1fb2	306	\<^v>
275	‼ Double Exclamation Point	P1 51 (0x33)	0x1fb3	307	\<!!>
276	¶ Paragraph Symbol	P1 52 (0x34)	0x1fb4	308	\<|P>
277	§ Section symbol	P1 53 (0x35)	0x1fb5	309	\<|S>
278	▬ Horizontal Solid Rectangle	P1 54 (0x36)	0x1fb6	310	\<#]>
279	↨ Underlined Bidirectional Vertical Arrow	P1 55 (0x37)	0x1fb7	311	\<-|>
280	↑ Up Arrow	P1 56 (0x38)	0x1fb8	312	\<|^>
281	↓ Down Arrow	P1 57 (0x39)	0x1fb9	313	\<|v>
282	→ Right Arrow	P1 58 (0x3a)	0x1fba	314	\<-}>
283	← Left Arrow	P1 59 (0x3b)	0x1fbb	315	\<{->
284	∟ Diagonally Flipped Logical Not	P1 60 (0x3c)	0x1fbc	316	\<`->
285	↔ Bidirectional Horizontal Arrow	P1 61 (0x3c)	0x1fbd	317	\<()>
286	▲ Solid Upward Triangle	P1 62 (0x3e)	0x1fbe	318	\<#^>
287	▼ Solid Downward Triangle	P1 63 (0x3f)	0x1fbf	319	\<#v>
288	ã a Tilde Small	P1 64 (0x40)	0x1fc0	320	\<a~>

Figure 5-7 (Part 1 of 7). Code Page P1

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
289	β	s Sharp Small	P1 65 (0x41)	0x1fc1	321	\<ss>
290	Â	a Circumflex Capital	P1 66 (0x42)	0x1fc2	322	\<A^>
291	À	a Grave Capital	P1 67 (0x43)	0x1fc3	323	\<A`>
292	Á	a Acute Capital	P1 68 (0x44)	0x1fc4	324	\<A'>
293	Ã	a Tilde Capital	P1 69 (0x45)	0x1fc5	325	\<A~>
294	ø	o Slash Small	P1 70 (0x46)	0x1fc6	326	\<o/>
295	Ê	e Circumflex Capital	P1 71 (0x47)	0x1fc7	327	\<E^>
296	Ë	e Umlaut Capital	P1 72 (0x48)	0x1fc8	328	\<E">
297	È	e Grave Capital	P1 73 (0x49)	0x1fc9	329	\<E`>
298	Í	i Acute Capital	P1 74 (0x4a)	0x1fca	330	\<I'>
299	Î	i Circumflex Capital	P1 75 (0x4b)	0x1fcb	331	\<I^>
300	Ï	i Umlaut Capital	P1 76 (0x4c)	0x1fcc	332	\<I">
301	Ì	i Grave Capital	P1 77 (0x4d)	0x1fcd	333	\<I`>
302	Ø	Slashed o Capital	P1 78 (0x4e)	0x1fce	334	\<O/>
303	ð	eth Icelandic Small	P1 79 (0x4f)	0x1fcf	335	\<d+>
304	ý	y Acute Small	P1 80 (0x50)	0x1fd0	336	\<y'>
305	þ	Thorn Icelandic Small	P1 81 (0x51)	0x1fd1	337	\<Ip>
306	¸	Cedilla Accent	P1 82 (0x52)	0x1fd2	338	\<_,>
307	¤	International Currency Symbol	P1 83 (0x53)	0x1fd3	339	\<o*>
308	Ð	eth Icelandic Capital	P1 84 (0x54)	0x1fd4	340	\<D+>
309	Ý	y Acute Capital	P1 85 (0x55)	0x1fd5	341	\<Y'>
310	Þ	Thorn Icelandic Capital	P1 86 (0x56)	0x1fd6	342	\<IP>
311	®	Registered Trademark Symbol	P1 87 (0x57)	0x1fd7	343	\<r0>
312	¾	Three Quarters	P1 88 (0x58)	0x1fd8	344	\<34>
313	¯	Overbar Accent, Macron Accent	P1 89 (0x59)	0x1fd9	345	\<-->
314	¨	Umlaut Accent	P1 90 (0x5a)	0x1fda	346	\<_">
315	´	Acute Accent	P1 91 (0x5b)	0x1fdb	347	\<_'>
316	‗	Double Underscore	P1 92 (0x5c)	0x1fdc	348	\<__>
317	õ	o Tilde Small	P1 93 (0x5d)	0x1fdd	349	\<o~>
318	ı	Small i Dotless	P1 94 (0x5e)	0x1fde	350	\<i>
319	Ô	o Circumflex Capital	P1 95 (0x5f)	0x1fdf	351	\<O^>
320	Ò	o Grave Capital	P1 96 (0x60)	0x1fe0	352	\<O`>
321	Ó	o Acute Capital	P1 97 (0x61)	0x1fe1	353	\<O'>

Figure 5-7 (Part 2 of 7). Code Page P1

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
322	Õ	o Tilde Capital	P1 98 (0x62)	0x1fe2	354	\<O~>
323	^{3}	Superscript 3	P1 99 (0x63)	0x1fe3	355	\<^3>
324	Û	u Circumflex Capital	P1 100 (0x64)	0x1fe4	356	\<U^>
325	Ù	u Grave Capital	P1 101 (0x65)	0x1fe5	357	\<U`>
326	Ú	u Acute Capital	P1 102 (0x66)	0x1fe6	358	\<U'>
327	ą	a Ogonek Small	P1 103 (0x67)	0x1fe7	359	\<a,>
328	ě	e Caron Small	P1 104 (0x68)	0x1fe8	360	\<ev>
329	č	c Caron Small	P1 105 (0x69)	0x1fe9	361	\<cv>
330	ć	c Acute Small	P1 106 (0x6a)	0x1fea	362	\<c'>
331	ę	e Ogonek Small	P1 107 (0x6b)	0x1feb	363	\<e,>
332	ů	u Overcircle Small	P1 108 (0x6c)	0x1fec	364	\<uo>
333	ď	d Caron Small	P1 109 (0x6d)	0x1fed	365	\<dv>
334	ĺ	l Acute Small	P1 110 (0x6e)	0x1fee	366	\<l'>
335	Ą	a Ogonek Capital	P1 111 (0x6f)	0x1fef	367	\<A,>
336	Ě	e Caron Capital	P1 112 (0x70)	0x1ff0	368	\<Ev>
337	Č	c Caron Capital	P1 113 (0x71)	0x1ff1	369	\<Cv>
338	Ć	c Acute Capital	P1 114 (0x72)	0x1ff2	370	\<C'>
339	ˇ	Caron Accent	P1 115 (0x73)	0x1ff3	371	\<_v>
340	Ę	e Ogonek Capital	P1 116 (0x74)	0x1ff4	372	\<E,>
341	Ů	u Overcircle Capital	P1 117 (0x75)	0x1ff5	373	\<Uo>
342	Ď	d Caron Capital	P1 118 (0x76)	0x1ff6	374	\<Dv>
343	Ĺ	l Acute Capital	P1 119 (0x77)	0x1ff7	375	\<L'>
344	ľ	l Caron Small	P1 120 (0x78)	0x1ff8	376	\<lv>
345	ň	n Caron Small	P1 121 (0x79)	0x1ff9	377	\<nv>
346	đ	d Stroke Small	P1 122 (0x7a)	0x1ffa	378	\<d->
347	ř	r Caron Small	P1 123 (0x7b)	0x1ffb	379	\<rv>
348	ś	s Acute Small	P1 124 (0x7c)	0x1ffc	380	\<s'>
349	°	Overcircle Accent	P1 125 (0x7d)	0x1ffd	381	\<_o>
350	ł	l Slash Small	P1 126 (0x7e)	0x1ffe	382	\<l->
351	ń	n Acute Small	P1 127 (0x7f)	0x1fff	383	\<n'>
352	š	s Caron Small	P1 128 (0x80)	0x1e80	384	\<sv>
353	Ľ	l Caron Capital	P1 129 (0x81)	0x1e81	385	\<Lv>
354	Ň	n Caron Capital	P1 130 (0x82)	0x1e82	386	\<Nv>

Figure 5-7 (Part 3 of 7). Code Page P1

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
355	Ř r Caron Capital	P1 131 (0x83)	0x1e83	387	\<Rv>
356	Ś s Acute Capital	P1 132 (0x84)	0x1e84	388	\<S'>
357	˙ Overdot Accent	P1 133 (0x85)	0x1e85	389	\<.>
358	ż z Overdot Small	P1 134 (0x86)	0x1e86	390	\<z.>
359	˛ Ogonek Accent	P1 135 (0x87)	0x1e87	391	\<,,>
360	Ż z Overdot Capital	P1 136 (0x88)	0x1e88	392	\<Z.>
361	ž z Caron Small	P1 137 (0x89)	0x1e89	393	\<zv>
362	ź z Acute Small	P1 138 (0x8a)	0x1e8a	394	\<z'>
363	Ž z Caron Capital	P1 139 (0x8b)	0x1e8b	395	\<Zv>
364	Ź z Acute Capital	P1 140 (0x8c)	0x1e8c	396	\<Z'>
365	Ł l Slash Capital	P1 141 (0x8d)	0x1e8d	397	\<L->
366	Ń n Acute Capital	P1 142 (0x8e)	0x1e8e	398	\<N'>
367	Š s Caron Capital	P1 143 (0x8f)	0x1e8f	399	\<Sv>
368	ť t Caron Small	P1 144 (0x90)	0x1e90	400	\<tv>
369	ŕ r Acute Small	P1 145 (0x91)	0x1e91	401	\<r'>
370	ő o Double Acute Small	P1 146 (0x92)	0x1e92	402	\<o=>
371	ű u Double Acute Small	P1 147 (0x93)	0x1e93	403	\<u=>
372	Ť t Caron Capital	P1 148 (0x94)	0x1e94	404	\<Tv>
373	Ŕ r Acute Capital	P1 149 (0x95)	0x1e95	405	\<R'>
374	Ő o Double Acute Capital	P1 150 (0x96)	0x1e96	406	\<O=>
375	Ű u Double Acute Capital	P1 151 (0x97)	0x1e97	407	\<U=>
376	ă a Breve Small	P1 152 (0x98)	0x1e98	408	\<au>
377	ğ g Breve Small	P1 153 (0x99)	0x1e99	409	\<gu>
378	İ i Overdot Capital	P1 154 (0x9a)	0x1e9a	410	\<I.>
379	Ă a Breve Capital	P1 155 (0x9b)	0x1e9b	411	\<Au>
380	Ğ g Breve Capital	P1 156 (0x9c)	0x1e9c	412	\<Gu>
381	˘ Breve Accent	P1 157 (0x9d)	0x1e9d	413	\<_u>
382	˝ Double Acute Accent	P1 158 (0x9e)	0x1e9e	414	\<_=>
383	ş s Cedilla Small	P1 159 (0x9f)	0x1e9f	415	\<s,>
384	ℓ Liter Symbol	P1 160 (0xa0)	0x1ea0	416	\<l>
385	’n High Comma n Small	P1 161 (0xa1)	0x1ea1	417	\<,n>
386	Ş s Cedilla Capital	P1 162 (0xa2)	0x1ea2	418	\<S,>
387	¯ Macron Accent	P1 163 (0xa3)	0x1ea3	419	\<_->

Figure 5-7 (Part 4 of 7). Code Page P1

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
388	ţ	t Cedilla Small	P1 164 (0xa4)	0x1ea4	420	\<t,>
389	Ţ	t Cedilla Capital	P1 165 (0xa5)	0x1ea5	421	\<T,>
390	ā	a Macron Small	P1 166 (0xa6)	0x1ea6	422	\<a->
391	Ā	a Macron Capital	P1 167 (0xa7)	0x1ea7	423	\<A->
392	ĉ	c Circumflex Small	P1 168 (0xa8)	0x1ea8	424	\<c^>
393	Ĉ	c Circumflex Capital	P1 169 (0xa9)	0x1ea9	425	\<C^>
394	ˋ	High Reverse Solidus	P1 170 (0xaa)	0x1eaa	426	\<\\>
395	ċ	c Overdot Small	P1 171 (0xab)	0x1eab	427	\<c.>
396	Ċ	c Overdot Capital	P1 172 (0xac)	0x1eac	428	\<C.>
397	ė	e Overdot Small	P1 173 (0xad)	0x1ead	429	\<e.>
398	Ė	e Overdot Capital	P1 174 (0xae)	0x1eae	430	\<E.>
399	ē	e Macron Small	P1 175 (0xaf)	0x1eaf	431	\<e->
400	Ē	e Macron Capital	P1 176 (0xb0)	0x1eb0	432	\<E->
401	ǵ	g Acute Small	P1 177 (0xb1)	0x1eb1	433	\<g'>
402	ĝ	g Circumflex Small	P1 178 (0xb2)	0x1eb2	434	\<g^>
403	Ĝ	g Circumflex Capital	P1 179 (0xb3)	0x1eb3	435	\<G^>
404	ġ	g Overdot Small	P1 180 (0xb4)	0x1eb4	436	\<g.>
405	Ġ	g Overdot Capital	P1 181 (0xb5)	0x1eb5	437	\<G.>
406	Ģ	g Cedilla Capital	P1 182 (0xb6)	0x1eb6	438	\<G,>
407	ĥ	h Circumflex Small	P1 183 (0xb7)	0x1eb7	439	\<h^>
408	Ĥ	h Circumflex Capital	P1 184 (0xb8)	0x1eb8	440	\<H^>
409	ħ	h Stroke Small	P1 185 (0xb9)	0x1eb9	441	\<h->
410	Ħ	h Stroke Capital	P1 186 (0xba)	0x1eba	442	\<H->
411	ĩ	i Tilde Small	P1 187 (0xbb)	0x1ebb	443	\<i~>
412	Ĩ	i Tilde Capital	P1 188 (0xbc)	0x1ebc	444	\<I~>
413	ī	i Macron Small	P1 189 (0xbd)	0x1ebd	445	\<i->
414	Ī	i Macron Capital	P1 190 (0xbe)	0x1ebe	446	\<I->
415	į	i Ogonek Small	P1 191 (0xbf)	0x1ebf	447	\<i,>
416	Į	i Ogonek Capital	P1 192 (0xc0)	0x1ec0	448	\<I,>
417	ĳ	ij Ligature Small	P1 193 (0xc1)	0x1ec1	449	\<ij>
418	Ĳ	IJ Ligature Capital	P1 194 (0xc2)	0x1ec2	450	\<IJ>
419	ĵ	j Circumflex Small	P1 195 (0xc3)	0x1ec3	451	\<j^>
420	Ĵ	j Circumflex Capital	P1 196 (0xc4)	0x1ec4	452	\<J^>

Figure 5-7 (Part 5 of 7). Code Page P1

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
421	ķ	k Cedilla Small	P1 197 (0xc5)	0x1ec5	453	\<k,>
422	Ķ	k Cedilla Capital	P1 198 (0xc6)	0x1ec6	454	\<K,>
423	ĸ	k Greenlandic Small	P1 199 (0xc7)	0x1ec7	455	\<k>
424	ļ	l Cedilla Small	P1 200 (0xc8)	0x1ec8	456	\<l,>
425	Ļ	l Cedilla Capital	P1 201 (0xc9)	0x1ec9	457	\<L,>
426	ŀ	l Middle Dot Small	P1 202 (0xca)	0x1eca	458	\<l.>
427	Ŀ	l Middle Dot Capital	P1 203 (0xcb)	0x1ecb	459	\<L.>
428	ņ	n Cedilla Small	P1 204 (0xcc)	0x1ecc	460	\<n,>
429	Ņ	n Cedilla Capital	P1 205 (0xcd)	0x1ecd	461	\<N,>
430	ŋ	n Eng Lapp Small	P1 206 (0xce)	0x1ece	462	\<nj>
431	Ŋ	n Eng Lapp Capital	P1 207 (0xcf)	0x1ecf	463	\<Nj>
432	ō	o Macron Small	P1 208 (0xd0)	0x1ed0	464	\<o->
433	Ō	o Macron Capital	P1 209 (0xd1)	0x1ed1	465	\<O->
434	œ	oe Ligature Small	P1 210 (0xd2)	0x1ed2	466	\<oe>
435	Œ	oe Ligature Capital	P1 211 (0xd3)	0x1ed3	467	\<OE>
436	ŗ	r Cedilla Small	P1 212 (0xd4)	0x1ed4	468	\<r,>
437	Ŗ	r Cedilla Capital	P1 213 (0xd5)	0x1ed5	469	\<R,>
438	ŝ	s Circumflex Small	P1 214 (0xd6)	0x1ed6	470	\<s^>
439	Ŝ	s Circumflex Capital	P1 215 (0xd7)	0x1ed7	471	\<S^>
440	ŧ	t Stroke Small	P1 216 (0xd8)	0x1ed8	472	\<t->
441	Ŧ	t Stroke Capital	P1 217 (0xd9)	0x1ed9	473	\<T->
442	ũ	u Tilde Small	P1 218 (0xda)	0x1eda	474	\<u~>
443	Ũ	u Tilde Capital	P1 219 (0xdb)	0x1edb	475	\<U~>
444	ŭ	u Breve Small	P1 220 (0xdc)	0x1edc	476	\<uu>
445	Ŭ	u Breve Capital	P1 221 (0xdd)	0x1edd	477	\<Uu>
446	ū	u Macron Small	P1 222 (0xde)	0x1ede	478	\<u->
447	Ū	u Macron Capital	P1 223 (0xdf)	0x1edf	479	\<U->
448	ų	u Ogonek Small	P1 224 (0xe0)	0x1ee0	480	\<u,>
449	Ų	u Ogonek Capital	P1 225 (0xe1)	0x1ee1	481	\<U,>
450	ŵ	w Circumflex Small	P1 226 (0xe2)	0x1ee2	482	\<w^>
451	Ŵ	w Circumflex Capital	P1 227 (0xe3)	0x1ee3	483	\<W^>
452	ŷ	y Circumflex Small	P1 228 (0xe4)	0x1ee4	484	\<y^>
453	Ŷ	y Circumflex Capital	P1 229 (0xe5)	0x1ee5	485	\<Y^>

Figure 5-7 (Part 6 of 7). Code Page P1

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
454	Ÿ	y Umlaut Capital	P1 230 (0xe6)	0x1ee6	486	\<Y">
455	©	Copyright Symbol	P1 231 (0xe7)	0x1ee7	487	\<c0>
456	¹	Superscript One	P1 232 (0xe8)	0x1ee8	488	\<^1>
457	™	Trademark Symbol	P1 233 (0xe9)	0x1ee9	489	\<tm>
458	⅛	One Eighth	P1 234 (0xea)	0x1eea	490	\<18>
459	⅜	Three Eights	P1 235 (0xeb)	0x1eeb	491	\<38>
460	⅝	Five Eighths	P1 236 (0xec)	0x1eec	492	\<58>
461	⅞	Seven Eighths	P1 237 (0xed)	0x1eed	493	\<78>
462	×	Multiplication Sign	P1 238 (0xee)	0x1eee	494	\<x>
463	’	Right Single Quote	P1 239 (0xef)	0x1eef	495	\<‘>
464	“	Left Double Quote	P1 240 (0xf0)	0x1ef0	496	\<“>
465	”	Right Double Quote	P1 241 (0xf1)	0x1ef1	497	\<''>
466	=	Equal Sign Superscript	P1 242 (0xf2)	0x1ef2	498	\<^=>
467	-	Minus Sign Superscript	P1 243 (0xf3)	0x1ef3	499	\<^->
468	+	Plus Sign Superscript	P1 244 (0xf4)	0x1ef4	500	\<^+>
469	∞	Infinity symbol Superscript	P1 245 (0xf5)	0x1ef5	501	\<8^>
470	π	Pi Symbol Superscript	P1 246 (0xf6)	0x1ef6	502	\<^p>
471	Δ	Delta Symbol Superscript	P1 247 (0xf7)	0x1ef7	503	\<^d>
472	→	Right Arrow Superscript	P1 248 (0xf8)	0x1ef8	504	\<^}>
473	/	Slash Superscript	P1 249 (0xf9)	0x1ef9	505	\<^/>
474	†	Dagger	P1 250 (0xfa)	0x1efa	506	\<|+>
475	<	Left Angle Superscript	P1 251 (0xfb)	0x1efb	507	\<^[>
476	>	Right Angle Superscript	P1 252 (0xfc)	0x1efc	508	\<^]>
477	℞	Prescription Symbol	P1 253 (0xfd)	0x1efd	509	\<Rx>
478	⋶	'Is Not An Element' Symbol	P1 254 (0xfe)	0x1efe	510	\<e/>
479	∴	'Therefore' Symbol	P1 255 (0xff)	0x1eff	511	\<:.>

Figure 5-7 (Part 7 of 7). Code Page P1

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
480	↗ Increase	P2 32 (0x20)	0x1da0	544	\</^>
481	↘ Decrease	P2 33 (0x21)	0x1da1	545	\<"v>
482	‡ Double Dagger	P2 34 (0x22)	0x1da2	546	\<++>
483	≠ Not Equal Symbol	P2 35 (0x23)	0x1da3	547	\</=>
484	∨ OR Symbol	P2 36 (0x24)	0x1da4	548	\<v>
485	∧ AND Symbol	P2 37 (0x25)	0x1da5	549	\<^>
486	∥ Parallel	P2 38 (0x26)	0x1da6	550	\<||>
487	∠ Angle Symbol	P2 39 (0x27)	0x1da7	551	\</_>
488	< Left Angle Bracket	P2 40 (0x28)	0x1da8	552	\<{>
489	> Right Angle Bracket	P2 41 (0x29)	0x1da9	553	\<}>
490	∓ Minus Or Plus Sign	P2 42 (0x2a)	0x1daa	554	\<-+>
491	⌑ Lozenge	P2 43 (0x2b)	0x1dab	555	\<{}>
492	′ Minutes Symbol	P2 44 (0x2c)	0x1dac	556	\<'>
493	∫ Integral Symbol	P2 45 (0x2d)	0x1dad	557	\<S>
494	∪ Union	P2 46 (0x2e)	0x1dae	558	\<u>
495	⊂ 'Is Included In' Symbol	P2 47 (0x2f)	0x1daf	559	\<(_>
496	⊃ 'Includes' Symbol	P2 48 (0x30)	0x1db0	560	\<_)>
497	⊕ Circle Plus, Closed Sum	P2 49 (0x31)	0x1db1	561	\<0+>
498	∟ Right Angle Symbol	P2 50 (0x32)	0x1db2	562	\<L>
499	⊗ Circle Multiply	P2 51 (0x33)	0x1db3	563	\<0x>
500	″ Seconds Symbol	P2 52 (0x34)	0x1db4	564	\<">
501	‗ Double Overline	P2 53 (0x35)	0x1db5	565	\<=^>
502	ψ Psi Small	P2 54 (0x36)	0x1db6	566	\<&y>
503	ε Epsilon Small	P2 55 (0x37)	0x1db7	567	\<&e>
504	λ Lambda Small	P2 56 (0x38)	0x1db8	568	\<&l>
505	η Eta Small	P2 57 (0x39)	0x1db9	569	\<&h>
506	ι Iota Small	P2 58 (0x3a)	0x1dba	570	\<&i>
507	⎛ Upper Left Parenthesis Section	P2 59 (0x3b)	0x1dbb	571	\<^(>
508	⎝ Lower Left Parenthesis Section	P2 60 (0x3c)	0x1dbc	572	\<v(>
509	‰ Permille Symbol	P2 61 (0x3d)	0x1dbd	573	\<%%>
510	θ Theta Small	P2 62 (0x3e)	0x1dbe	574	\<&&>
511	κ Kappa Small	P2 63 (0x3f)	0x1dbf	575	\<&k>
512	ω Omega Small	P2 64 (0x40)	0x1dc0	576	\<&w>

Figure 5-8 (Part 1 of 5). Code Page P2

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
513	ν	Nu Small	P2 65 (0x41)	0x1dc1	577	\<&n>
514	ο	Omicron Small	P2 66 (0x42)	0x1dc2	578	\<&o>
515	ρ	Rho Small	P2 67 (0x43)	0x1dc3	579	\<&r>
516	γ	Gamma Small	P2 68 (0x44)	0x1dc4	580	\<&g>
517	θ	Theta Small	P2 69 (0x45)	0x1dc5	581	\<&q>
518	⎞	Upper Right Parenthesis Section	P2 70 (0x46)	0x1dc6	582	\<^)>
519	⎠	Lower Right Parenthesis Section	P2 71 (0x47)	0x1dc7	583	\<v)>
520	≅	'Congruent To' Symbol	P2 72 (0x48)	0x1dc8	584	\<~=>
521	ξ	Xi Small	P2 73 (0x49)	0x1dc9	585	\<&x>
522	χ	Chi Small	P2 74 (0x4a)	0x1dca	586	\<&c>
523	υ	Upsilon Small	P2 75 (0x4b)	0x1dcb	587	\<&u>
524	ζ	Zeta Small	P2 76 (0x4c)	0x1dcc	588	\<&z>
525	⌠	Lower Right/Upper Left Brace Section	P2 77 (0x4d)	0x1dcd	589	\<|'>
526	ʅ	Upper Right/Lower Left Brace Section	P2 78 (0x4e)	0x1dce	590	\<`|>
527	$_0$	Zero Subscript	P2 79 (0x4f)	0x1dcf	591	\<v0>
528	$_1$	One Subscript	P2 80 (0x50)	0x1dd0	592	\<v1>
529	$_2$	Two Subscript	P2 81 (0x51)	0x1dd1	593	\<v2>
530	$_3$	Three Subscript	P2 82 (0x52)	0x1dd2	594	\<v3>
531	$_4$	Four Subscript	P2 83 (0x53)	0x1dd3	595	\<v4>
532	$_5$	Five Subscript	P2 84 (0x54)	0x1dd4	596	\<v5>
533	$_6$	Six Subscript	P2 85 (0x55)	0x1dd5	597	\<v6>
534	$_7$	Seven Subscript	P2 86 (0x56)	0x1dd6	598	\<v7>
535	$_8$	Eight Subscript	P2 87 (0x57)	0x1dd7	599	\<v8>
536	$_9$	Nine Subscript	P2 88 (0x58)	0x1dd8	600	\<v9>
537	⊥	Perpendicular	P2 89 (0x59)	0x1dd9	601	\<|_>
538	⍚	Total Symbol	P2 90 (0x5a)	0x1dda	602	\<-^>
539	Ψ	Psi Capital	P2 91 (0x5b)	0x1ddb	603	\<&Y>
540	Π	Pi Capital	P2 92 (0x5c)	0x1ddc	604	\<&P>
541	Λ	Lambda Capital	P2 93 (0x5d)	0x1ddd	605	\<&L>
542	●	Bottle Symbol	P2 94 (0x5e)	0x1dde	606	\<db>
543	ƀ	Substitute Blank	P2 95 (0x5f)	0x1ddf	607	\<b/>
544	∂	Partial Differential Symbol	P2 96 (0x60)	0x1de0	608	\<d>
545	∿	Sine Symbol	P2 97 (0x61)	0x1de1	609	\<~~>

Figure 5-8 (Part 2 of 5). Code Page P2

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
546	□ Open Square	P2 98 (0x62)	0x1de2	610	\<[->
547	■ Solid Square	P2 99 (0x63)	0x1de3	611	\<[#>
548	▨ Slash Square	P2 100 (0x64)	0x1de4	612	\<[/>
549	⎲ Upper Summation Section	P2 101 (0x65)	0x1de5	613	\<^S>
550	⎳ Lower Summation Section	P2 102 (0x66)	0x1de6	614	\<vS>
551	Ξ Xi Capital	P2 103 (0x67)	0x1de7	615	\<&X>
552	∝ 'Proportional To' Symbol	P2 104 (0x68)	0x1de8	616	\<o(>
553	Δ Delta Capital	P2 105 (0x69)	0x1de9	617	\<&D>
554	ϒ Upsilon Capital	P2 106 (0x6a)	0x1dea	618	\<&U>
555	≃ 'Approximately Equal To' Symbol	P2 107 (0x6b)	0x1deb	619	\<~->
556	∼ Cycle Symbol, 'Equivalent To' Symbol	P2 108 (0x6c)	0x1dec	620	\<~>
557	^{0} Zero Superscript	P2 109 (0x6d)	0x1ded	621	\<^0>
558	^{4} Four Superscript	P2 110 (0x6e)	0x1dee	622	\<^4>
559	^{5} Five Superscript	P2 111 (0x6f)	0x1def	623	\<^5>
560	^{6} Six Superscript	P2 112 (0x70)	0x1df0	624	\<^6>
561	^{7} Seven Superscript	P2 113 (0x71)	0x1df1	625	\<^7>
562	^{8} Eight Superscript	P2 114 (0x72)	0x1df2	626	\<^8>
563	^{9} Nine Superscript	P2 115 (0x73)	0x1df3	627	\<^9>
564	Ø Zero Slash	P2 116 (0x74)	0x1df4	628	\<0/>
565	₧ Paseta Sign	P2 117 (0x75)	0x1df5	629	\<Pt>
566	⌐ Flipped Logical Not	P2 118 (0x76)	0x1df6	630	\<.->
567	╡ Right Side Middle - Double Horizontal	P2 119 (0x77)	0x1df7	631	\<H6>
568	╢ Right Side Middle - Double Vertical	P2 120 (0x78)	0x1df8	632	\<V6>
569	╖ Upper Right Corner - Double Vertical	P2 121 (0x79)	0x1df9	633	\<V9>
570	╕ Upper Right Corner - Double Hor.	P2 122 (0x7a)	0x1dfa	634	\<H9>
571	╜ Lower Right Corner - Double Vertical	P2 123 (0x7b)	0x1dfb	635	\<V3>
572	╛ Lower Right Corner - Double Hor.	P2 124 (0x7c)	0x1dfc	636	\<H3>
573	╞ Left Side Middle - Double Horizontal	P2 125 (0x7d)	0x1dfd	637	\<H4>
574	╟ Left Side Middle - Double Vertical	P2 126 (0x7e)	0x1dfe	638	\<V4>
575	╧ Bottom Side Middle - Double Horizontal	P2 127 (0x7f)	0x1dff	639	\<H2>
576	╨ Bottom Side Middle - Double Vertical	P2 128 (0x80)	0x1c80	640	\<V2>
577	╤ Top Side Middle - Double Horizontal	P2 129 (0x81)	0x1c81	641	\<H8>
578	╙ Lower Left Corner - Double Vertical	P2 130 (0x82)	0x1c82	642	\<V1>

Figure 5-8 (Part 3 of 5). Code Page P2

Font Position	Character		Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
579	╘	Lower Left Corner - Double Hor.	P2 131 (0x83)	0x1c83	643	\<H1>
580	╒	Upper Left Corner - Double Hor.	P2 132 (0x84)	0x1c84	644	\<H7>
581	╓	Upper Left Corner - Double Vertical	P2 133 (0x85)	0x1c85	645	\<V7>
582	╫	Intersection - Double Vertical	P2 134 (0x86)	0x1c86	646	\<V5>
583	╪	Intersection - Double Horizontal	P2 135 (0x87)	0x1c87	647	\<H5>
584	▌	Bright Character Cell - Left Half	P2 136 (0x88)	0x1c88	648	\<B4>
585	▐	Bright Character Cell - Right Half	P2 137 (0x89)	0x3c89	649	\<B6>
586	α	Alpha Small	P2 138 (0x8a)	0x1c8a	650	\<&a>
587	β	Beta Small	P2 139 (0x8b)	0x1c8b	651	\<&b>
588	Γ	Gamma Capital	P2 140 (0x8c)	0x1c8c	652	\<&G>
589	π	Pi Small	P2 141 (0x8d)	0x1c8d	653	\<&p>
590	Σ	Sigma Capital/Summation Sign	P2 142 (0x8e)	0x1c8e	654	\<&S>
591	σ	Sigma Small	P2 143 (0x8f)	0x1c8f	655	\<&s>
592	τ	Tau Small	P2 144 (0x90)	0x1c90	656	\<&t>
593	Φ	Phi Capital	P2 145 (0x91)	0x1c91	657	\<&F>
594	Θ	Theta Capital	P2 146 (0x92)	0x1c92	658	\<&Q>
595	Ω	Omega Capital/Ohm Sign	P2 147 (0x93)	0x1c93	659	\<&W>
596	δ	Delta Small	P2 148 (0x94)	0x1c94	660	\<&d>
597	∞	Infinity	P2 149 (0x95)	0x1c95	661	\<8>
598	φ	Phi Small	P2 150 (0x96)	0x1c96	662	\<&f>
599	∈	'Is An Element Of' Symbol	P2 151 (0x97)	0x1c97	663	\<e>
600	∩	Intersection	P2 152 (0x98)	0x1c98	664	\<n>
601	≡	Identity Symbol	P2 153 (0x99)	0x1c99	665	\<==>
602	≥	'Greater Than or Equal To' Symbol	P2 154 (0x9a)	0x1c9a	666	\<}=>
603	≤	'Less Than or Equal To' Symbol	P2 155 (0x9b)	0x1c9b	667	\<{=>
604	⌠	Upper Integral Section	P2 156 (0x9c)	0x1c9c	668	\<^I>
605	⌡	Lower Integral Section	P2 157 (0x9d)	0x1c9d	669	\<vI>
606	≈	Double Equivalent	P2 158 (0x9e)	0x1c9e	670	\<=~>
607	•	Solid Overcircle	P2 159 (0x9f)	0x1c9f	671	\<#o>
608	√	Radical Symbol, Square Root	P2 160 (0xa0)	0x1ca0	672	\<v->
609	╥	Top Side Middle - Double Vertical	P2 161 (0xa1)	0x1ca1	673	\<V8>
610	ⁿ	Superscript n	P2 162 (0xa2)	0x1ca2	674	\<^n>
611		Numeric Space	P2 163 (0xa3)	0x1ca3	675	\<0->

Figure 5-8 (Part 4 of 5). Code Page P2

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
612	℄ Center Line	P2 164 (0xa4)	0x1ca4	676	\<-)>
613	⌴ Counter Bore	P2 165 (0xa5)	0x1ca5	677	\<-u>
614	⌵ Counter Sink	P2 166 (0xa6)	0x1ca6	678	\<Iv>
615	↧ Depth	P2 167 (0xa7)	0x1ca7	679	\<'v>
616	⌀ Diameter	P2 168 (0xa8)	0x1ca8	680	\<-=>

Figure 5-8 (Part 5 of 5). Code Page P2

Related Information

In this book: "conv" on page 3-39, "NLchar" on page 3-276, "NLescstr, NLunescstr, NLflatstr" on page 3-278, "fonts" on page 4-68, "data stream" on page 5-5, "hft" on page 6-23, and "keyboard" on page 6-78.

Keyboard Description and Character Reference.

ebcdic

Purpose

Maps the EBCDIC character set.

Synopsis

cat /usr/pub/ebcdic

Description

In the following table columns correspond to the high-order hexadecimal digits and rows correspond to low-order hexadecimal digits. The cells contain equivalent hexadecimal ASCII values, the symbols, and mnemonics common to EBCDIC and ASCII. Exceptions are flagged in the following table by (1) through (8):

	0_	1_	2_	3_	4_	5_	6_	7_
_0	00 nul	10 dle	80 ds	90	20 sp	26 &	2D -	BA
_1	01 soh	11 dc1	81 sos	91	A0	A9	2F /	BB
_2	02 stx	12 dc2	82 fs	16 syn	A1	AA	B2	BC
_3	03 etx	13 (1)	83	93	A2	AB	B3	BD
_4	9C pf	9D res	84 byp	94 pn	A3	AC	B4	BE
_5	09 ht	85 nl	0A lf	95 rs	A4	AD	B5	BF
_6	86 lc	08 bs	17 etb	96 uc	A5	AE	B6	C0
_7	7F del	87 il	1B esc	04 eot	A6	AF	B7	C1
_8	97	18 can	88	98	A7	B0	B8	C2
_9	8D	19 em	89	99	A8	B1	B9	60 '
_A	8E smm	92 cc	8A sm	9A	D5	21 !	CB	3A :
_B	0B vt	8F cu1	8B cu2	9B cu3	2E .	24 $	2C ,	23 #
_C	0C ff	1C (2)	8C	14 dc4	3C <	2A *	25 %	40 @
_D	0D cr	1D (3)	05 enq	15 nak	28 (	29)	5F _	27 '
_E	0E so	1E (4)	06 ack	9E	2B +	3B ;	3E >	3D =
_F	0F si	1F (5)	07 bel	1A sub	7C |(6)	7E ~(7)	3F ?	22 "

Figure 5-9 (Part 1 of 2). EBCDIC Character Set

	8_	9_	A_	B_	C_	D_	E_	F_
_0	C3	CA	D1	D8	7B {	7D }	5C \	30 0
_1	61 a	6A j	E5	D9	41 A	4A J	9F	31 1
_2	62 b	6B k	73 s	DA	42 B	4B K	53 S	32 2
_3	63 c	6C l	74 t	DB	43 C	4C L	54 T	33 3
_4	64 d	6D m	75 u	DC	44 D	4D M	55 U	34 4
_5	65 e	6E n	76 v	DD	45 E	4E N	56 V	35 5
_6	66 f	6F o	77 w	DE	46 F	4F O	57 W	36 6
_7	67 g	70 p	78 x	DF	47 G	50 P	58 X	37 7
_8	68 h	71 q	79 y	E0	48 H	51 Q	59 Y	38 8
_9	69 i	72 r	7A z	E1	49 I	52 R	5A Z	39 9
_A	C4	5E ^(8)	D2	E2	E8	EE	F4	FA
_B	C5	CC	D3	E3	E9	EF	F5	FB
_C	C6	CD	D4	E4	EA	F0	F6	FC
_D	C7	CE	5B [	5D]	EB	F1	F7	FD
_E	C8	CF	D6	E6	EC	F2	F8	FE
_F	C9	D0	D7	E7	ED	F3	F9	FF

Figure 5-9 (Part 2 of 2). EBCDIC Character Set

	EBCDIC	ASCII
(1)	13 tm	13 dc3
(2)	1C ifs	1C fs
(3)	1D igs	1D gs
(4)	1E irs	1E rs
(5)	1F ius	1F us
(6)	4F |	7C |
(7)	5F ¬	7E ~
(8)	9A	5E ^

File

/usr/pub/ebcdic

Related Information

The **dd** command in *AIX Operating System Commands Reference.*

environment

Purpose

Describes the user environment.

Synopsis

Basic Environment

HOME = *path name of home directory*
PATH = *directory search sequence*
TERM = *terminal type*
TZ = *time zone information*

International Character Support Environment

NLFILE = *path name of environment file*
NLCTAB = *path name of collating tables*
NLLANG = *language name*

NLCURSYM = *currency symbol*
NLNUMSEP = *triad and decimal separators*

NLLDAY = *long day names*
NLLMONTH = *long month names*
NLSDAY = *short day names*
NLSMONTH = *short month names*
NLTMISC = *miscellaneous time strings*
NLTSTRS = *relative time names*
NLTUNITS = *time unit names*

NLDATE = *short date format*
NLLDATE = *long date format*
NLTIME = *time format*

Description

When a new process begins, the **exec** system call makes an array of strings available that have the form *name=value*. This array of strings is called the ***environment***. Each *name* defined by one of the strings is called an ***environment variable*** or ***shell variable***.

When using the **sh** command interpreter, additional names can be placed in the environment with the **export** or **env** command, or by adding a *name=value* prefix to any other command. See the **sh** command in *AIX Operating System Commands Reference* for more information about setting environment variables with shell commands.

Within a program, the **getenv** subroutine can be used to search the environment for the value of a given variable. The **exec** system call allows the entire environment to be set at one time, usually for a newly started child process.

When creating new environment variables, assure that their names do not conflict with those of standard variables used by the shell and other programs, such as **MAIL**, **PS1**, **PS2**, and **IFS**.

The Basic Environment

When you log in, a number of environment variables are automatically set by the system before running your login profile, **.profile**. These variables make up the ***basic environment***:

HOME The full path name of the user's login or ***home*** directory. The **login** program sets this to the name specified in the **/etc/passwd** file.

PATH The sequence of directories that commands such as **sh**, **time**, **nice**, and **nohup** search when looking for a command whose path name is incomplete. The directory names are separated by colons. **PATH** is set by the system login profile, **/etc/profile**.

TERM The type of terminal for which output is to be prepared. Commands such as **mm** and **tplot** use this information to manipulate special capabilities, if any, of that terminal. The **curses**, **extended curses**, and **terminfo** subroutines also use the value of **TERM**. For asynchronous terminals, **TERM** is set by the **getty** command to a value defined in **/etc/ports**. For the RT PC console, **TERM** is set using the **termdef** subroutine.

TZ Time zone information. **TZ** is set in the system login profile, **/etc/profile**.

The fields of **TZ** are separated by colons. The first field has three subfields, *lcl*, *n*, and *dst*. If they are not supplied, the defaults for the U.S.A. Eastern Standard (and Daylight Savings) Time Zone are used.

The additional fields of **TZ** specify when daylight savings time begins and ends. If these fields are not supplied, U.S.A. rules for daylight savings time apply. Daylight savings time specifications apply to all years, and may require adjustment from year to year in locations where the start of daylight savings time varies.

TZ is represented in the format:

lclndst[:*bgn*:*end*:*chgwd*:*chghr*:*chgamt*]

where

lcl Is the standard local time zone abbreviation. This name must be nine bytes or fewer, and cannot contain periods, colons, or hyphens. To be compatibile with other operating systems, this name should be three bytes.

n Is the difference in local time from GMT in hours (a number from -12 to 12), optionally followed by a period and a number of minutes. Negative differences are for locations east of Greenwich. To be compatibile with other operating systems, this difference should not contain minutes.

dst Is the abbreviation for the local daylight savings time zone, if any. This name must be nine bytes or fewer, and cannot contain periods, colons, or hyphens. To be compatibile with other operating systems, this name should be three bytes long.

bgn Is the beginning day of daylight savings time, if any. This number is the Julian value, or number of days into the year. The value is specified for a non-leap year and adjusted as necessary for leap years.

end Is the ending day (Julian) of daylight savings time, if any. If the value of *end* is smaller than the value of *bgn*, daylight savings time crosses the new year, as is the practice in the Southern Hemisphere.

chgwd Is the weekday of the change to daylight savings time, if any. Values range from 1 to 7, with 1 representing Monday and 7 representing Sunday. This value specifies that daylight savings time begins and ends on the first named weekday *before* the Julian dates specified by *bgn* and *end*. (For example, you could specify the last Saturday in October.) If the value of this field is 0 or not entered, the exact Julian date given is used (corrected for leap year where necessary).

chghr Is the hour of the change to daylight savings time, if any (number of elapsed hours in the day, optionally followed by a period and a number of minutes).

chgamt Is the amount of the change to daylight savings time. This value is specified by an optionally signed number of minutes, optionally followed by a period and a number of minutes. That is, [-]*hh*[.*mm*].

For example, a **TZ** string for Lord Howe Island, Australia in 1985-86 might be:

```
AusLHIst-10.30AusLHIdt:300:60::2:0.30
```

International Character Support Environment

A special set of environment variables defines the international character support configuration. These environment variables locate configuration information and tailor input and output forms of dates, times, and monetary sums according to "national" or local requirements. If an environment variable *value* for international character support contains blanks, the value appear in quotes and blanks cannot separate the equals sign from the variable name or the value.

Environment variables for international character support are specified in the process environment using ordinary shell environment variables, or in the text file whose path name is specified by the shell environment variable **NLFILE**. Values specified in the process environment take precedence over values specified in **NLFILE**. If a given environment variable is not set either in the process environment or in **NLFILE**, or if a specified value is the null string, a default value is used.

The **NLgetenv** subroutine provides a program with a method to retrieve a value associated with an international character support environment variable.

Environment variables that establish the local environment vocabulary specification consist of a sequence of strings separated by colons. The set of conventions being used is identified by the value of **NLLANG**. Each string must be a translation of the U.S. English name or symbol used in the defaults, in exactly the same order.

The **NLDATE**, **NLLDATE**, and **NLTIME** variables are format strings that can be specified as simple format strings or as **NLstrtime** format strings. These strings are arbitrary, but can not begin with an * (asterisk). When the patterns listed in the following table appear in a simple format string, **NLgetenv** substitutes the appropriate part of the date or time.·

Pattern	Meaning	Replacement
DD	Numeric day of the month	**%d**
MM	Numeric month	**%m**
YY	Numeric year (two digits)	**%y**
YYYY	Numeric year (four digits)	**%Y**
mon	Month specified by **NLSMONTH**	**%h**
month	Month specified by **NLSMONTH**	**%lh**
hh	Numeric hour	**%H**
mm	Numeric minutes	**%M**
ss	Numeric seconds	**%S**
aa	Numeric AM/PM indicator	**%p**

Characters that are not part of replacement patterns are not translated. These are some examples of simple format strings:

```
mon DD, YYYY          hh.mm
MM/DD/YY              hh:mm:ss
DD.MM.YY              hh:mm aa
YYYY-MM-DD
DD month YY
```

Format strings of the style of **NLstrtime** follow the same form as the *format* parameter of **NLstrtime**, except that the string must be preceded by an asterisk and it cannot contain the formats **%D**, **%sD**, **%lD**, **%T**, **%sT**, or **%r**. The asterisk is not translated and does not become part of the result.

The environment variables are described as follows:

NLCTAB The path name of the file containing tables that define the current collating sequence, as produced by the **ctab** command. The default path name is:

```
/etc/nls/ctab/default
```

NLCURSYM The currency symbol name and placement. The default value is:

```
:$:L:
```

NLDATE The environment format string specifying the short form of the date. This format is used by **NLstrtime** when the format **%D** is encountered. The default is:

```
MM/DD/YY
```

NLFILE The path name of a file containing other environment variable definitions for international character support. **NLFILE** cannot be defined within a file that is identifed by another **NLFILE** definition. There is no default path name.

NLLANG The environment language label for the set of variables and environment format strings used for language conventions. The default value is:

```
u.s.english
```

NLLDATE The environment format string specifying the long form of the date. This form is used by **NLstrtime** when the formats **%lD** or **%sD** are encountered. The default long date format string is:

```
mon DD, YYYY
```

NLLDAY The full (long) names for the days of the week. The default value is:

```
Sunday:Monday:Tuesday:Wednesday:Thursday:Friday:Saturday
```

NLLMONTH The full (long) names for the months of the year. The default value is:

```
January:February:March:April:May:June:July:\
August:September:October:November:December
```

NLNUMSEP The numeric triad and decimal separators. The first of the two separators is the triad separator, which is used to separate groups of three digits in decimal values. The default value for **NLNUMSEP** is:

```
:,:.:
```

NLSDAY The short names of the days of the week. Names should be the same length, and of 5 or fewer characters. The default short name string is:

```
Sun:Mon:Tue:Wed:Thu:Fri:Sat
```

NLSMONTH The short names of the months of the year. Names should be the same length, and of 5 or fewer characters. The default value is:

```
Jan:Feb:Mar:Apr:May:Jun:Jul:Aug:Sep:Oct:Nov:Dec
```

NLTIME The environment format string specifying the format of the time, that is used by **NLstrtime** when the formats **%T**, **%sT**, or **%r** are encountered. The default time format string is:

```
hh:mm:ss
```

NLTMISC Miscellaneous strings needed for input and output of date and time specifications. The default miscellaneous string value is:

```
at:each:every:on:through:am:pm
```

NLTSTRS The relative or informal names needed for input of date and time specifications to the **remind** and **at** commands (see the **remind** and **at** commands in *AIX Operating System Commands Reference*). The default informal time string value is:

```
now:yesterday:tomorrow:noon:midnight:next:weekdays:weekend
```

NLTUNITS The singular and plural forms for all names of units of time, used for input of date specifications to the **at** command. The default string value for units of time is:

```
minute:minutes:hour:hours:day:days:week:weeks:month:months:year:years
```

Files

/etc/environment	Sets the basic environment for all processes.
/etc/profile	Allows variables to be added to the environment by the shell.
$HOME/.profile	Sets the environment for a specific user's needs.
/etc/nls/ctab/default	Sets the international character support environment.

Related Information

In this book: "exec: execl, execv, execle, execve, execlp, execvp" on page 2-34, "getenv, NLgetenv" on page 3-208, "NLstrtime" on page 3-288, "NLtmtime" on page 3-291, "termdef" on page 3-352, "passwd" on page 4-112, "profile" on page 4-127, and "TERM" on page 5-72.

The **ctab**, **env**, **export**, **login**, and **sh** commands in *AIX Operating System Commands Reference.*

"Overview of International Character Support" in *IBM RT PC Managing the AIX Operating System.*

eqnchar

Purpose

Identifies special character definitions for **eqn** and **neqn** formatters.

Synopsis

eqn /usr/pub/eqnchar [files] | **troff** [options]
neqn /usr/pub/eqnchar [files] | **nroff** [options]

Description

The **eqnchar** file contains **troff** and **nroff** character definitions used to construct special scientific symbols. These definitions are primarily intended to be used with the **eqn** and **neqn** formatters. The **eqnchar** file contains definitions for the following characters:

ciplus	⊕	||	‖	square	□
citimes	⊗	langle	⟨	circle	○
wig	~	rangle	⟩	blot	■
-wig	≃	ppd	⊥	bullet	●
>wig	≳	hbar	ℏ	prop	∝
<wig	≲	<->	↔	empty	∅
=wig	≅	<=>	⟺	member	∈
star	∗	|<	≮	nomem	∉
bigstar	✱	|>	≯	cup	∪
=dot	≐	ang	∠	cap	∩
orsign	∨	rang	∟	incl	⊑
andsign	∧	3dot	⋮	subset	⊂
=del	≜	thf	∴	supset	⊃
oppA	∀	quarter	1/4	!subset	⊆
oppE	∃	3quarter	3/4	!supset	⊇
angstrom	Å	degree	°	scrL	ℓ
==<	≦	==>	≧		

Figure 5-10. The eqnchar Characters

File

/usr/pub/eqnchar

Related Information

The **eqn, nroff, and troff** commands in *AIX Operating System Commands Reference.*

fcntl.h

Purpose

Defines file control options.

Synopsis

#include <fcntl.h>

Description

The **fcntl.h** header file defines the values that can be specified for the *cmd* and *arg* parameters of the **fcntl** system call, and for the *oflag* parameter of the **open** system call.

```
/* Flag values accessible to open and fcntl */
/* The first three can only be set by open */

#define O_RDONLY  0
#define O_WRONLY  1
#define O_RDWR    2
#define O_NDELAY  04   /* Non-blocking I/O */
#define O_APPEND  010  /* (0x08) append (writes guaranteed */
                                    /*        at the end) */

/* Flag values accessible only to open */
#define O_CREAT 00400  /* (0x0100) open with create (uses third */
                                   /*                open arg) */
#define O_TRUNC 01000  /* (0x0200) open with truncation */
#define O_EXCL  02000  /* (0x0400) exclusive open */

/* fcntl requests */
#define F_DUPFD 0      /* Duplicate fildes */
#define F_GETFD 1      /* Get fildes flags */
#define F_SETFD 2      /* Set fildes flags */
#define F_GETFL 3      /* Get file flags */
#define F_SETFL 4      /* Set file flags */
```

```
#define F_GETLK 5      /* Get file lock */
#define F_SETLK 6      /* Set file lock */
#define F_SETLKW 7     /* Set file lock and wait */

/* file segment locking set data type - information passed to */
/* system by user */

struct flock {
     short   l_type;
     short   l_whence;
     long    l_start;
     long    l_len;      /* len = 0 means until end of file */
     unsigned long   l_sysid;
     short   l_pid;
};

/* file segment locking types */
#define F_RDLCK 01     /* Read lock */
#define F_WRLCK 02     /* Write lock */
#define F_UNLCK 03     /* Remove lock(s) */
```

File

/usr/include/fcntl.h

Related Information

In this book: "fcntl" on page 2-44 and "open" on page 2-90.

fullstat.h

Purpose

Defines the data structure returned by the **fullstat** system call.

Synopsis

#include <sys/fullstat.h>

Description

The **fullstat.h** header file defines the data structure that is returned by the **fullstat** and **ffullstat** system calls. This file also defines the *cmd* arguments that are used by **fullstat** and **ffullstat**.

```
struct fullstat
{
   /* Beginning of stat block replica... */
   dev_t   st_dev;          /* ID of device containing */
                            /* a directory entry for this file */
                            /* File serial + device uniquely */
                            /* identifies the file within the system */
   ino_t   st_ino;          /* File serial number */
   ushort  st_mode;         /* File mode; see #defines below */
   short   st_nlink;        /* Number of links to file */
   ushort  st_uid;          /* User ID of the owner of the file */
   ushort  st_gid;          /* Group ID of the file group */
   dev_t   st_rdev;         /* ID of this device */
                            /* This entry is defined only for */
                            /* character or block special files */
   off_t   st_size;         /* File size in bytes */
   time_t  st_atime;        /* Time of last access */
   time_t  st_mtime;        /* Time of last data modification */
   time_t  st_ctime;        /* Time of last file status change */
                            /* Time measured in seconds since */
                            /* 00:00:00 GMT, Jan. 1, 1970 */
   /* ...End of stat block replica */
   ushort  fst_uid_raw;     /* Untranslated uid of the file */
   ushort  fst_gid_raw;     /* Untranslated gid of the file */
   vtype   fst_type;        /* Vnode type */
   tagtype fst_uid_rev_tag;     /* uid translation tag */
   tagtype fst_gid_rev_tag;     /* gid translation tag */
   short * fst_other_gid_list;  /* Pointer to first group ID on */
                                /*   alternate concurrent group list */
   short   fst_other_gid_count; /* Number of group IDs on */
                                /*   alternate concurrent group list */
   long    fst_vfs;         /* Virtual file system ID */
   long    fst_nid;         /* Node id where the file resides */
   int     fst_flag;        /* Indicates whether directory or */
                            /* file is a virtual mount point */
   long    fst_i_gen;       /* Inode generation number */
   long    fst_reserved[8]; /* Reserved */
};
   /* Defines for fullstat or ffullstat cmd argument */
   #define FL_STAT         0x0      /* Fullstat */
   #define FL_STAT_REV     0x1      /* Fullstat with uid/gid */
                                   /*   reverse mapping     */
   #define FL_STAT_OTHER   0x2      /* Reverse mapping, "biased" */
                                   /*   toward another uid/gid    */
   /* Defines to tell whether a file or directory is mounted upon */
   #define FS_VMP          0x1      /* Virtual mount point  */
```

File

/usr/include/sys/fullstat.h

Related Information

In this book: "fullstat, ffullstat" on page 2-50.2 and "types.h" on page 5-75.

greek

Purpose

Maps Greek characters.

Purpose

cat /usr/pub/greek [| **greek -T***terminal*]

Description

The **/usr/pub/greek** file shows the mapping from ASCII characters to the “shift-out” graphics in effect between **SO** and **SI** on TELETYPE Model 37 work stations equipped with an extended (128) character set. These codes are the default Greek characters produced by the **nroff** command. Use the **greek** command to translate these characters for display on other work stations. The file contains:

```
alpha    α  A  |  beta      β  B  |  gamma   γ    \
GAMMA    Γ  G  |  delta     δ  D  |  DELTA   Δ    W
epsilon  ε  S  |  zeta      ζ  Q  |  eta     η    N
THETA    Θ  T  |  theta     θ  O  |  lambda  λ    L
LAMBDA   Λ  E  |  mu        μ  M  |  nu      ν    @
xi       ξ  X  |  pi        π  J  |  PI      Π    P
rho      ρ  K  |  sigma     σ  Y  |  SIGMA   Σ    R
tau      τ  I  |  phi       φ  U  |  PHI     Φ    F
psi      ψ  V  |  PSI       Ψ  H  |  omega   ω    C
OMEGA    Ω  Z  |  nabla     ∇  [  |  not     ¬    _
partial  ∂  ]  |  integral  ∫  ^
```

Figure 5-2. Greek Characters

File

/usr/pub/greek

Related Information

The **300**, **4014**, **450**, **greek**, **hp**, **nroff**, **tc**, and **troff** commands in *AIX Operating System Commands Reference.*

hosts

Purpose

Defines *hostname* and associated addresses for hosts in the network.

Synopsis

/etc/hosts/

Description

This file contains the *hostnames* and their addresses for hosts in the network. This file is used to resolve a name into an address (that is, to translate a *hostname* into its Internet address).

This file can contain three additional entries (reserved, ***well-known*** host names):

nameserver
timeserver
printserver

If a *hostname* is not in the **hosts** file, a request to resolve *hostname* to an address is sent to another host, the host associated with the **nameserver** entry. Generally, most hosts in the network have a **nameserver** entry. For example, in a small network, one host can run the **nameserver** daemon; the **/etc/hosts/** file on that host contains an entry for all hosts in the network. Each of the other hosts in the network contains an entry for itself and an entry for the **nameserver**.

Note: A **nameserver** entry must point to a foreign host.

The host associated with **timeserver** responds to **setclock** requests (a means for synchronizing the time among hosts in the network). Each host may or may not run **timeserver**. If network time is to be used on a particular host, that host must have a **timeserver** entry in its **/etc/hosts/** file. The **printserver** entry identifies the default host for receiving print requests.

To tailor the network environment for a particular host, modify its **/etc/hosts/** file. Each entry is of the form:

address hostname hostname hostname hostname

where *address* can be specified in decimal or octal and *hostname* is a string with a maximum length of 24 characters and no embedded blanks. Multiple *hostnames* (or aliases) can be specified as long as the total number of characters does not exceed 100 characters; the entry must be contained on one line.

Examples

Following are sample entries in the **/etc/hosts/** files for three different hosts in a network:

Host 1

```
192.9.200.1  host1 host1a host1b
192.9.200.2  host2
192.9.200.3  host3
128.114.1.15
128.114.1.14
128.114.2.7
192.9.200.2  timeserver
192.9.200.3  printserver
```

Host 2

```
192.9.200.2  host2
192.9.200.1  nameserver
192.9.200.2  timeserver
192.9.200.3  printserver
```

Host 3

```
192.9.200.3  host3
192.9.200.1  nameserver
192.9.200.2  timeserver
```

In this sample network, the **/etc/hosts/** file for host1 contains address entries for all hosts in the network; host1 runs the **nameserver** daemon. (The **/etc/hosts/** file of host1 can contain a **nameserver** entry if the entry specifies some host other than host1.) The entries in the hostl **/etc/hosts/** file that begin with 128.114 indicate that hostl also resolves names for hosts on more than one network. hostl is also known (aliased) as hostla and hostlb.

The **/etc/hosts/** file of host2 contains an address entry only for host2 itself; **host2** runs the **timeserver** daemon. The **/etc/hosts/** file of host3 contains an address entry only for host3 itself. All three hosts use hostl to perform the **nameserver** function and host2 to perform the **timeserver** function. **host3** runs the **printserver** daemon and receives remote print requests from **hostl** and **host2**.

File

/etc/hosts	host name data base.

Related Information

In this book: "gethostbyaddr, gethostbyname, sethostent, endhostent" on page 8-13.

math.h

Purpose

Defines math subroutines and constants.

Synopsis

#include <math.h>

Description

This header file contains declarations of all the subroutines in the Math Library (**libm.a**) and of various subroutines in the Standard C Library (**libc.a**) that return floating-point values.

It defines the structure and constants for the **matherr** error-handling mechanism used by the math subroutines. (See "matherr" on page 3-238 for details about this mechanism.)

Among other things, **math.h** defines the following constant, which is used as an error-return value:

HUGE The maximum value of a single-precision floating-point number.

If you define the **_C_func** preprocessor variable before including **math.h**, then **math.h** defines macros that make the names of certain math subroutines appear to the compiler as **_C_***xxxx*. The following names are redefined to have a **_C_** prefix:

exp
log
log10
sqrt
sin
cos
tan
asin
acos
atan
atan2

These special names instruct the C compiler to generate code that avoids the overhead of the math library subroutines and issues compatible-mode floating-point calls directly. See "fpfp" on page 3-170 for information about compatible mode.

The following mathematical constants are also defined for your convenience:

M_E The base of natural logarithms (*e*)

M_LOG2E The base-2 logarithm of *e* ($\log_2 e$)

M_LOG10E	The base-10 logarithm of *e* ($\log_{10} e$)
M_LN2	The natural logarithm of 2 ($\log_e 2$)
M_LN10	The natural logarithm of 10 ($\log_e 10$)
M_PI	π, the ratio of the circumference of a circle to its diameter
M_PI_2	The value of $\pi \div 2$
M_PI_4	The value of $\pi \div 4$
M_1_PI	The value of $1 \div \pi$
M_2_PI	The value of $2 \div \pi$
M_2_SQRTPI	The value of 2 divided by the positive square root of π
M_SQRT2	The positive square root of 2
M_SQRT1_2	The positive square root of ½.

The **math.h** file contains an **#include** statement that imbeds another header file named **values.h**. This header file defines a number of machine-dependent constants, and it is discussed on page 5-77.

Files

/usr/include/math.h
/usr/include/values.h

Related Information

In this book: "matherr" on page 3-238 and "values.h" on page 5-77.

mm

Purpose

Provides the **mm** macro package for formatting documents.

Synopsis

mm [options] [files]
nroff -mm [options] [files]
nroff -cm [options] [files]
mmt [options] [files]
troff -mm [options] [files]

Description

This package provides a formatting capability for a very wide variety of documents. How a document is typed and edited on the system is independent of whether the document is to be eventually formatted at a terminal or photoset. See the following references for further details.

Files

/usr/lib/tmac/tmac.m
/usr/lib/tmac/sys.name
/usr/lib/macros/mm[nt]

Related Information

The **mm**, **mmt**, **nroff**, and **troff** commands in *AIX Operating System Commands Reference.*

mptx

Purpose

Provides the macro package for formatting a permuted index.

Synopsis

nroff -mptx [*flag* . . .] [*file* . . .]
troff -mptx [*flag* . . .] [*file* . . .]

Description

This package provides a definition for the **.xx** macro which is used for formatting a permuted index produced by the **ptx** program. This package does not provide any other formatting capabilities such as headers and footers. Use this macro package in conjunction with the **mm** macro package for these or other capabilities. In this case, the **-mptx** flag must follow the **-mm** flag. For example:

```
nroff -mm -mptx file
```

or

```
mm -mptx file
```

Files

/usr/lib/tmac/tmac.ptx
/usr/lib/macros/ptx

Related Information

In this book: “mm” on page 5-62.

The **mm, nroff, ptx, troff** commands in *AIX Operating System Commands Reference.*

mv

Purpose

Provides a **troff** macro package for typesetting view graphs and slides.

Synopsis

mvt [**-a**] [**-rw1**] [*flag* . . .] [*file* . . .]
troff [**-a**] [**-rw1**] [**-rX1**] **-mv** [*flag* . . .] [*file* . . .]

Description

This package makes it easy to typeset view graphs and projection slides in a variety of sizes. A few macros (briefly described in the following) accomplish most of the formatting tasks needed in making transparencies. The facilities of **troff, cw, eqn**, and **tbl** are available for more difficult tasks.

The output can be previewed on most terminals. To preview on some devices, specify the **-rX1** option (this option is automatically specified by the **mvt** command, when that command is invoked with certain options). To preview output on other terminals, specify the **-a** option. The **-rw1** option suppresses the printing of cross-hairs and crop marks.

The available macros are:

.A [*x*] Places text that follows at the first indentation level (left margin); the presence of *x* suppresses the 1/2 line spacing from the preceding text.

.B [*m*[*s*]]
Places text that follows at the second indentation level. Text is preceded by a mark. *m* is the mark, the default is a large bullet. *s* is the increment or decrement to the point size of the mark with respect to the prevailing point size. The default is 0. If *s* is 100, it causes the point size of the mark to be the same as that of the default mark.

.BX *str1* [*str2*] [*f*]
Encloses *str1* in a box and appends *str2* (if any) to it. *str1* is set in the prevailing font unless *f* names a different font.

.C [*m* [*s*]]
Same as **.B**, but for the third indentation level. The default mark is a dash.

.CN [*args*]
Ends a constant-width font display.

.CW [*args*]
Begins a constant-width font display at the current indentation level.

.D [*m* [*s*]]
Same as **.B**, but for the fourth indentation level. The default mark is a small bullet.

.DF *n f* [*n f* . . .]
Defines font positions. This may not appear within a foil's input text (for example, it may only appear after all the input text for a foil, but before the next foil-start macro). *n* is the position of font *f*, up to four "*n f*" pairs can be specified. The first font named becomes the prevailing font. The initial setting is (**H** is a synonym for **G**):

```
.DF 1 H 2 I 3 B 4 S
```

.DV [*a*] [*b*] [*c*] [*d*] [*e*]
Alters the vertical spacing between indentation levels. The *a, b, c,* and *d* values alter the spacing for **.A, .B, .C**, and **.D** respectively. The *e* value is the pre-spacing and post-spacing for constant-width font displays bracketed by the **.CW** and **.CN** macros. Arguments that are not null must have dimensions. Null arguments leave the corresponding spacing unaffected. Initial setting is:

```
.DV .5v .5v .5v 0v .5v
```

.I [*in*] [*a* [*x*]]
Changes the current text indent, but does not affect titles. *in* is the indent in inches, unless dimensioned. The default is 0. If *in* is signed, it is an increment or decrement. The presence of *a* invokes the **.A** macro and passes *x*, if any, to it.

.S [*p*] [*l*] Sets the point size and line length. *p* is the point size, the default is previous. If *p* is 100, the point size reverts to the *initial* default for the current foil-start macro. If *p* is signed, it is an increment or decrement. The default is 18 for **.VS, .VH**, and **.SH**, and 14 for the other foil-start macros. *l* is the line length in inches unless dimensioned. The default is 4.2 inches for **.Vh**, 3.8 inches for **.Sh**, 5 inches for **.SH**, and 6 inches for the other foil-start macros).

.Sh [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 5 × 7 inches.

.SH [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 7 × 9 inches.

.Sw [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 7 × 5 inches.

.SW [*n*] [*i*] [*d*]
Same as **.VS**, except that foils size is 7 × 5.4 inches.

.T *string* Prints *string* as an over-size, centered title.

.U *str1* [*str2*]
Underlines *str1* and concatenates *str2* (if any) to it.

.Vh [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 5 × 7 inches.

.VH [*n*] [*i*] [*d*]
Same as **.VS**, except that foils size is 7 × 9 inches.

.VS [*n*] [*i*] [*d*]
Foil-start macro; foil size is to be 7 × 7 inches. *n* is the foil number, *i* is the foil identification, *d* is the date. The foil-start macro resets all parameters (indent, point size, and so on) to initial default values, except for the values of *i* and *d* arguments that came from a previous foil-start macro; it also invokes the **.A** macro.

The naming convention for this and the eight other foil-start macros is that the first character of the name (**V** or **S**) distinguishes between view graphs and slides, respectively, while the second character indicates whether the foil is square (**S**), small wide (**w**), small high (**h**), big wide (**W**), or big high (**H**). Slides are thinner than the corresponding view graphs. For slides, the ratio of the longer dimension to the shorter one is larger than for view graphs. As a result, slide foils can be used for view graphs, but not the opposite. Alternately, view graphs can accommodate a bit more text.

.Vw [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 7 inches wide × 5 inches high.

.VW [*n.*] [*i*] [*d*]
Same as **.VS**, except that foil size is 7 × 5.4 inches.

.WS [*w*] [*string*]
Reserves *w* amount of white space. *w* must have dimensions. If *string* is present, prints, in the reserved space, the caption:

Paste up *string* here.

The **.S, .DF**, **.DV, .U** and **.BX** macros do not cause a break. The **.I** macro causes a break only if it is invoked with more than one argument. All the other macros cause a break.

The macro package also recognizes the following upper case synonyms for the corresponding lower case **troff** requests:

.AD .BR .CE .HY .NA .NH .NX .SO .SP .TA .TI

The **Tm** string produces the trademark symbol.

The ~ (tilde) character is translated into a blank on output.

The following **troff** symbols are defined:

***t**	The ASCII tab character.
***E**	The ellipsis (. . .). Do not use this symbol within constant-width text.
***u**	The short name of the operating system in small capital letters.
***(UU**	The short name of the operating system with a leading full-cap letter.
***(UF**	The full name of the operating system.
***(Tm**	The trademark symbol.

Note: The VW and SW foils are meant to be 9 inches wide by 7 inches high. However, the typesetter paper is generally only 8 inches wide, so they are printed 7 inches wide by 5.4 inches high. They need to be enlarged by a factor of 9/7 before they can be used as view graphs.

Files

/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca

Related Information

The **cw**, **eqn**, **mmt**, **tbl**, **troff** commands in *AIX Operating System Commands Reference.*

networks

Purpose

Contains the network name data base.

Synopsis

/etc/networks

Description

The **networks** file contains information about the known networks that comprise the DARPA Internet. Each network is represented by a single line in the **networks** file. The format for the entries in **networks** is:

name number aliases

where:

name is the official network name.

number is the network number.

aliases are the unofficial names used for the network.

Items on a line are separated by one or more blanks or tab characters. Comments begin with the # character, and routines that search **networks** do not interpret characters from the beginning of a comment to the end of that line. Network numbers are specified in Internet dot notation. A network name can contain any printable character except a field delimiter, new line character, or comment character.

The **networks** file is normally created from the official network data base maintained at the Network Information Control Center (NIC). The file may need to be modified locally to include unofficial aliases or unknown networks.

File

/etc/networks	Network name data base.

Related Information

In this book: "getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent" on page 8-20.

param.h

Purpose

Describes system parameters.

Synopsis

#include <sys/param.h>

Description

Parameters vary among systems using the AIX operating system. For the RT PC, these parameters are in this file. The most significant parameters are:

BSIZE Indicates the kernel buffer size. RT PC has a buffer size of 2048 bytes. This determines the size of block clusters on a file system. Since the size of a block is 512 bytes, a cluster is 4 blocks.

NOFILE Indicates the maximum open file allowed per process. This value is 200.

NCARGS Indicates the maximum number of characters, including terminating NULs that may be passed using the **exec** system call.

File

/usr/include/sys/param.h

protocols

Purpose

Contains the protocol name data base.

Synopsis

/etc/protocols

Description

The **protocols** file contains information about the known protocols used in the DARPA Internet. Each protocol is represented by a single line in the **protocols** file. The format for the entries in **protocols** is:

name number aliases

where:

name is the official protocol name.

number is the protocol number.

aliases are the unofficial names used for the protocol.

Items on a line are separated by one or more blanks or tab characters. Comments begin with the # character, and routines that search **protocols** do not interpret characters from the beginning of a comment to the end of that line. A protocol name can contain any printable character except a field delimiter, new line character, or comment character.

File

/etc/protocols	Protocol name data base.

Related Information

In this book: "getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent" on page 8-24.

resolv.conf

Purpose

Contains name server and domain name information.

Synopsis

/etc/resolv.conf

Description

The resolver configuration file (**resolv.conf**) contains nameserver and domain information.

Generally, the **resolv.conf** file is used when the name server resides on a remote system. Typically, the only name server to be queried is on the local system and the domain name is retrieved from the system.

If the **resolv.conf** file exists, the resolver routines in **gethostbyname** and **gethostbyaddr** use an RFC883 domain name server. If this file does not exist, an IEN116 nameserver, whose address must be defined in the **/etc/hosts** file under the name **nameserver**, is used.

The configuration options are:

nameserver *address*
where *address* is the Internet address (in dot notation) of a name server the resolver subroutines should query. The file should contain at least one name server entry; up to **MAXNS** name servers may be listed.

If no name servers are listed, the name server on the local system is used. If more than one name server is listed, the resolver routines query each entry listed, repeating until the query succeeds or the maximum numbers of attempts have been made.

domain *name*
where *name* is the default domain name to append to names that do not have a dot in them.

If there are no domain entries, the domain given by the **gethostname** subroutine (everything following the first dot) is used. If the host name does not contain a domain part, the root domain is assumed.

Each line in the **resolv.conf** file must start with either **nameserver** or **domain**, followed by blanks or tabs and a corresponding *address* or *name*.

File

/etc/resolv.conf	Contains name server and domain name information.

Related Information

In this book: "gethostbyaddr, gethostbyname, sethostent, endhostent" on page 8-13, and "gethostname, sethostname" on page 8-18

services

Purpose

Contains the service name data base.

Synopsis

/etc/services

Description

The **services** file contains information about the known services used in the DARPA Internet. Each service is represented by a single line in the **services** file. The format for the entries in **services** is:

servname portnum/protname aliases

where:

servname is the official service name.

portnum/protname is the port number of the named service, separated from the name of the protocol by a / (slash).

aliases are the unofficial names used for the service.

Items on a line are separated by one or more blanks or tab characters. Comments begin with the # character, and routines that search **services** do not interpret characters from the beginning of a comment to the end of that line. A service name can contain any printable character except a field delimiter, new line character, or comment character.

A sample line in this file might look like:

```
chargen      19/tcp          ttytest source
```

File

/etc/services	Service name data base.

Related Information

In this book: "getservent, getservbyname, getservbyport, setservent, endservent" on page 8-26.

stat.h

Purpose

Defines the data structure returned by the **stat** system call.

Synopsis

#include <sys/stat.h>

Description

The **stat** and **fstat** system calls obtain information about a file that has a name. These system calls return a data structure defined by this include file. This file also defines encoding of the **st_mode** field. Note that in the structure below the octal value is shown. The hexadecimal equivalent values are also shown in parentheses.

```
struct  stat
{
    dev_t   st_dev;       /* ID of device containing */
                          /* a directory entry for this file */
                          /* File serial + device uniquely */
                          /* identifies the file within the system */
    ino_t   st_ino;       /* File serial number */
    ushort  st_mode;      /* File mode; see #defines below */
    short   st_nlink;     /* Number of links to file */
    ushort  st_uid;       /* User ID of the owner of the file */
    ushort  st_gid;       /* Group ID of the file group */
    dev_t   st_rdev;      /* ID of this device */
                          /* This entry is defined only for */
                          /* character or block special files */
    off_t   st_size;      /* File size in bytes */
    time_t  st_atime;     /* Time of the last access */
    time_t  st_mtime;     /* Time of the last data modification */
    time_t  st_ctime;     /* Time measured in seconds since */
                          /* 00:00:00 GMT, Jan. 1, 1970 */
};
```

```
#define S_IFMT   0170000          /* (0xF000) type of file */
#define S_IFDIR  0040000          /* (0x4000) directory */
#define S_ISDIR(m)  (((m) & (S_IFMT)) == (S_IFDIR))
#define S_IFCHR  0020000          /* (0x2000) character special */
#define S_ISCHR(m)  (((m) & (S_IFMT)) == (S_IFCHR))
#define S_IFBLK  0060000        /* (0x6000) block special  */
#define S_ISBLK(m)  (((m) & (S_IFMT)) == (S_IFBLK))
#define S_IFREG  0100000        /* (0x8000) regular */
#define S_ISREG(m)  (((m) & (S_IFMT)) == (S_IFREG))
#define S_IFIFO  0010000        /* (0x1000) fifo */
#define S_ISFIFO(m)  (((m) & (S_IFMT)) == (S_IFIFO))
#define S_ISUID  04000    /* (0x0800) set user id on execution */
#define S_ISGID  02000    /* (0x0400) set group id on execution */
#define S_ISVTX  01000    /* (0x0200) save swapped text even after use */
#define S_IRWXU  00700    /* (0x01C0) owner read,write,execute permission */
#define S_IREAD  00400    /* (0x0100) owner read permission */
#define S_IRUSR  00400    /* (0x0100) read permission, owner */
#define S_IWRITE 00200    /* (0x0080) owner write permission */
#define S_IWUSR  00200    /* (0x0080) owner write permission */
#define S_IEXEC  00100    /* (0x0040) owner execute/search permission */
#define S_IXUSR  00100    /* (0x0040) owner execute/search permission */
#define S_IRWXG  00070    /* (0x0038) group read,write,execute permission */
#define S_IRGRP  00040    /* (0x0020) group read permission */
#define S_IWGRP  00020    /* (0x0010) group write permission */
#define S_IXGRP  00010    /* (0x0008) group execute/search permission */
#define S_IRWXO  00007    /* (0x0007) other read,write,execute, permission */
#define S_IROTH  00004    /* (0x0004) other read permission */
#define S_IWOTH  00002    /* (0x0002) other write permission */
#define S_IXOTH  00001    /* (0x0001) other execute/search permission */
#define S_IFMPX S_IFCHR|S_ISVTX /* multiplex character special file */
#define S_ISMPX(m) (((m) & (S_IFMT|S_ISVTX)) == (S_IFMPX))
#define S_ENFMT S_ISGID       /* record locking enforcement flag */
```

Examples

The **S_IREAD**, **S_IWRITE**, and **S_IEXEC masks** can be used to test permissions in any of the three groups (owner, groups, or other) by shifting them. For example, to test for read access by group, use:

```
st_mode & (S_IREAD >> 3)
```

To test for global write access, use:

```
st_mode & (S_IWRITE >> 6)
```

File

/usr/include/sys/stat.h

Related Information

In this book: "stat, fstat" on page 2-159 and "types.h" on page 5-75.

TERM

Purpose

Lists conventional names for terminals.

Description

These names are used primarily for commands such as **mm** and **nroff**. These names are maintained as part of the shell environment in the variable **TERM**. See the **sh** command in *AIX Operating System Commands Reference* for an explanation of the shell. Also see "profile" on page 4-127 and "environment" on page 5-47 in this book for use of the **TERM** environment variable.

TERM	**Terminal Description**
ibm3161	IBM 3161 ASCII Display
ibm3161	IBM 3163 ASCII Display
ibm3161-C	IBM 3161 ASCII Display with cartridge (for international character support)
ibm3162	IBM 3162 ASCII Display (for international character support)
ibm5081	IBM 5081 Color Display
ibm5151	IBM Monochrome Display and Printer Adapter with IBM Personal Computer Display
ibm5154	IBM PC Enhanced Graphics Adapter with IBM Personal Computer Display
ibm5154	IBM PC Enhanced Graphics Adapter with IBM Personal Computer Enhanced Color Display
ibm5154	IBM Advanced Color Graphics Display Adapter with IBM Advanced Color Graphics Display
ibm6153	IBM RT PC Advanced Monochrome Graphics Display Adapter with IBM RT PC Advanced Monochrome Graphics Display
ibm6153-40	IBM 6153 terminal using a 40-column font
ibm6153-90	IBM 6153 terminal using a 90-column font
ibm6154	IBM 6154 terminal
ibm6154-40	IBM 6154 terminal using a 40-column font
ibm6154-90	IBM 6154 terminal using a 90-column font
ibm6155	IBM 6155 terminal
ibm6155-56	IBM 6155 terminal using a 56-column font
ibm6155-113	IBM 6155 terminal using a 113-column font
vt100	DEC VT100[1]

[1] Trademark of Digital Equipment Corporation.

vt220	DEC VT220[1]
dumb	Terminal types with no special features (such as reverse line motion) (implies **-c**)
lp	Line Printer (implies **-c**) (must pipe through **lpr** or some such filter)
37	Teletype Model 37 KSR
42	ADM 42 (implies **-c**)
300	DASI (DTC, GSI) 300
300s	DASI 300s
300-12	DASI 300 at 12-pitch
300s-12	DASI 300s at 12-pitch
tn300	TermiNet 300 (implies **-c**)
382	DTC 382
450	DASI 450 (same as Diablo 1620) DEFAULT
450-12	DASI 450 (same as Diablo 1620) 12-pitch
2631	HP 2631 series line printer (implies **-c**)
2631-e	HP 2631 series (expanded mode) (implies **-c**)
2631-c	HP 2631 series (compressed mode) (implies **-c**)
4000a	Trendata 4000a

Up to eight characters chosen from `[-a-z0-9]` make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a - (hyphen). Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept parameters such as **-T***term* where *term* is one of the names in the preceding list. If the parameter is not in the list, the commands should obtain the terminal type from the environment variable TERM, which in turn should contain *term*. Any unknown terminal is treated as a **dumb** terminal.

This list does not include all supported terminals. See “terminfo” on page 4-148 for additional TERM variables.

File

/user/lib/help/term

Related Information

In this book: "environment" on page 5-47 and "terminfo" on page 4-148.

The **mm, sh, stty, tabs, nroff**, and **environ** commands in *AIX Operating System Commands Reference.*

types.h

Purpose

Defines primitive system data types.

Synopsis

#include <sys/types.h>

Description

The data types defined in this include file are used in the RT PC system source code. Some data of these types are accessible to user code:

```
typedef struct {int r[1];} * physadr;
typedef long                 level_t
typedef long                 daddr_t;
typedef char *               caddr_t;
typedef unsigned int         uint;
typedef unsigned short       ushort;
typedef unsigned long        ulong;
typedef ushort               ino_t;
typedef short                cnt_t;
typedef long                 time_t;
typedef int                  label_t[11];
typedef int                  dev_t;
typedef long                 off_t;
typedef long                 paddr_t;
typedef long                 key_t;
```

Notes:

daddr_t This data type is used for disk addresses, except in i-nodes on disk. See the "fs" on page 4-74 for the format of disk addresses used in i-nodes.

time_t Times are encoded in seconds since 00:00:00 GMT, January 1, 1970.

dev_t The major and minor parts of a device code specify kind of device and unit number of the device, and they depend on the system customization.

off_t Offsets are measured in bytes from the beginning of a file.

label_t Variables of this type are used to save the processor state while another process is running.

File

/usr/include/sys/types.h

Related Information

In this book: "fs" on page 4-74 and "values.h" on page 5-77.

values.h

Purpose

Defines machine-dependent values.

Synopsis

#include <values.h>

Description

This header file contains a set of manifest constants that are conditionally defined for particular processor architectures. The model for integers is assumed to be a ones- or twos-complement binary representation, in which the sign is represented by the value of the high-order bit.

BITS(*type*)	The number of bits in the specified data type
HIBITS	A short integer with only the high-order bit set (0x8000)
HIBITL	A long integer with only the high-order bit set (0x80000000)
HIBITI	A regular integer with only the high-order bit set (the same as **HIBITL**)
MAXSHORT	The maximum value of a signed short integer (0x7FFF ≡ 32767)
MAXLONG	The maximum value of a signed long integer (0x7FFFFFFF ≡ 2147483647)
MAXINT	The maximum value of a signed regular integer (the same as **MAXLONG**)
MAXFLOAT	The maximum value of a single-precision floating-point number
MAXDOUBLE	The maximum value of a double-precision floating-point number
LN_MAXDOUBLE	The natural logarithm of **MAXDOUBLE**
MINFLOAT	The minimum positive value of a single-precision floating-point number
MINDOUBLE	The minimum positive value of a double-precision floating-point number

FSIGNIF The number of significant bits in the mantissa of a single-precision floating-point number

DSIGNIF The number of significant bits in the mantissa of a double-precision floating-point number

FMAXEXP The maxium exponent of a single-precision floating-point number

DMAXEXP The maxium exponent of a double-precision floating-point number

FMINEXP The minimum exponent of a single-precision floating-point number

DMINEXP The minimum exponent of a double-precision floating-point number

FMAXPOWTWO The largest power of two that can be exactly represented as a single-precision floating-point number

DMAXPOWTWO The largest power of two that can be exactly represented as a double-precision floating-point number.

File

/usr/include/values.h

Related Information

In this book: "math.h" on page 5-60, "types.h" on page 5-75.

Chapter 6. Special Files

About This Chapter

This chapter describes various special files that refer to specific hardware peripherals and RT PC system device drivers. The names of the entries are generally derived from names for the hardware as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding RT PC system device driver are discussed where applicable.

asy

Purpose

Supports the asynchronous adapter.

Description

The **asy** driver supports asynchronous ports. If a port is not installed, an attempt to open it fails. Each port can be individually programmed for speed (50-19.2K baud), character length, and parity. Output speed is always the same as input speed. The behavior of each adapter is described in the **termio** file.

The asynchronous port is a character-at-a-time device for both input and output. This characteristic limits the bandwidth, which can be achieved over a line and increases the interrupt loading on the central processor.

If the port was opened with the modem control bit present in the minor device (see the following text), modem control is enabled. If enabled, the driver waits in the **open** routine until data carrier detect is present. Once opened, if data carrier detect drops, the driver returns errors on any subsequent user read or write attempts of the asynchronous port. If the port was opened as a controlling teletype, a SIGHUP signal is generated to the process that performed the open.

Minor Device Numbers

The asynchronous ports are character devices. The low-order bit of the minor device number corresponds to the primary or secondary asynchronous ports. Bit 6 enables modem control on the selected port. Thus, minor device 0 corresponds to the first asynchronous port with modem control disabled, while minor device 65 corresponds to the second asynchronous port with modem control enabled.

Files

/dev/tty* for remote devices.
/dev/ltty* for local devices.

Related Information

In this book: "termio" on page 6-114 and "signal" on page 2-145.

The **config** command in *AIX Operating System Commands Reference.*

bus

Purpose

Supports the hardware bus interface.

Synopsis

#include <sys/hwdbus.h>
#include <fcntl.h>

Description

The **bus** file consists of a pseudo driver in the AIX kernel which allows a user program to enable the hardware I/O bus such that the address space can be addressed directly by the program rather than performing I/O through the AIX system calls.

The user program first opens the special file name associated with the device driver. Only **O_RDONLY**, **O_WRONLY**, and **O_RDWR** are valid values to be used with the **open** system call. Before the open, any addressing of I/O space generates a SIGSEGV signal.

Once the device driver is open, the user program should issue an **ioctl** system call to obtain the base addresses of the bus I/O and bus memory spaces. These base addresses are added to the appropriate address offsets to obtain an absolute I/O address.

ioctl Operations

The **ioctl** system call is invoked as follows:

```
ioctl (fildes, command, ptr)  /* get addresses */
int fildes;                   /* file descriptor */
int command;                  /* valid value is HWDBASE */
struct hwdbase                /* contains base addresses */
{
   char   *hwdio;
   char   *hwdmem;
} *ptr
```

The **hwdio** field is the address of the start of I/O memory allocated to the I/O port address. The **hwdmem** field is the address of the start of I/O memory allocated to dedicated memory, such as display refresh memory.

File

/dev/bus

Related Information

In this book: “hft” on page 6-23.

config

Purpose

Configures system device drivers.

Synopsis

#include <sys/types.h>
#include <sys/kcfg.h>
#include <sys/ksvc.h>

Description

The **config** driver is used to customize the Virtual Resource Manager (VRM) and the AIX kernel. Use of this pseudo-device is restricted to a user with superuser authority. Most operations are performed via **ioctl** system calls. The **write** system calls are accepted only in certain situations as described.

ioctl Operations

A list of **ioctl** calls along with the descriptions follows. These calls return a value of 0 upon successful completion. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

If an error occurred while issuing a supervisor call (SVC) to the VRM, **errno** is set to EIO. The VRM status code is obtained using the CFRSTAT type **ioctl** call. In all the following calls that pass a structure address, the call is aborted if the structure is not entirely within memory that is addressable by the calling process. In that case, **errno** is set to EFAULT.

CFBUFF Allocates and initializes the Block I/O Communication Area (BIOCA) in kernel memory. The *arg* parameter is a pointer to a **defdev** structure. (See **CFDDEV**, following, for the definition of this structure.)

CFDCODE Issues a **Define_Code** SVC to the VRM. The parameter is the following structure:

```
struct defcode {
    int  iocn;   /* IOCN to define */
    int  opts;   /* options word */
    int  ciocn;  /* IOCN to copy */
};
```

The options available are:

ADD_IOCN Add IOCN option

DEL_IOCN Delete IOCN option

DUP_IOCN Duplicate IOCN option.

If the option specifies deleting or duplicating a module, the action is performed immediately. To add a module, a single **write** system call must immediately follow with the contents of the **a.out** module. The VRM uses the **write** buffer directly. The **write** buffer is cleared when it is returned from a call. The module must be aligned on a 2K page boundary and not mixed with other data.

CFDDEV Configures a device in the VRM that is not a disk partition. The parameter is a pointer to the following structure:

```
struct defdev {
  unsigned short  iodn;  /* IODN to use */
  unsigned short  iocn;  /* IOCN to use */
  unsigned short  opts;  /* add/delete */
  unsigned short  chars; /* device characteristics */
  char  name[4];  /* device name */
  int  spare;
  union {
     struct {
           int  offhc;     /* offset to hdw characteristics */
           int  offdc;     /* offset to dev characteristics */
           int  offras;     /* offset to RAS info */
     } offsets
     int  ddi[1];  /* device dependent info */
} ddi_data;
```

The options available are:

ADD_IODN Add IODN option

DEL_IODN Delete IODN option.

A **Define_Device** SVC is issued to the VRM using the given data. The value is the Virtual Machine Interface (VMI) return code.

CFQDEV Issues a **Query_Device** SVC to the VRM. The parameter is a pointer to a structure of the following form:

```
struct qdev {
   unsigned short iodn;     /* IODN to query */
   unsigned short options;  /* Query device options */
   int   length;            /* length of the following */
   char  buffer[];     /* info returned here */
};
```

The options available are:

Q_HRDW Query hardware information

Q_DEV Query device information

Q_RAS Query RAS information.

After verifying the address and length of the structure, a **Query_Device** SVC is issued to the VRM with the structure given. The value returned is the value returned from the **Query_Device** SVC to the VRM.

CFRSTAT Returns the status code from the last SVC to the VRM issued from this driver. The parameter is ignored.

CFUDRV Configures a AIX device driver. The parameter is a pointer to the following structure:

```
struct unxdrv {
    dev_t  devno;  /* major/minor device number */
    unsigned short  iodn;   /* IODN to set */
    unsigned short  ddilen; /* device dependent info byte length
    unsigned short  lev;    /* interrupt level */
    union  {     /* optional device dependent info */
       . . .
    struct {     /* for Send_Command calls */
            int rv2,
            rv3,
            rv4,
            rv5,
            rv6,
    } ddi_sc;
  } ddi;
};
```

The device initialization routine for the given driver is called with the given **iodn** and device-dependent information. Some device drivers may interpret the **lev** field as the VMI interrupt level to use. (Sublevels are always assigned dynamically by the kernel.) The value of the CFUDRV type **ioctl** system call is the value returned by the device driver initialization routine. A successful call returns a value of 0, otherwise it returns a value of -1.

The device driver may modify the device-dependent information that is copied back into the structure provided by the caller of the **config** driver.

If both the **ddilen** and **iodn** fields are 0, the device initialization routine turns off the given minor number so that future calls to open that device will fail. If the **iodn** field is 0 and the **ddilen** field is a value other than 0, the device driver may perform various operations not directly relating to the minor device specified in the **devno** field. Device drivers associated with device managers issue a **Send_Command** SVC to the VRM device manager with the given values **rv2, rv3, rv4, rv5**, and **rv6** in registers 2, 3, 4, 5, and 6. The driver waits for the manager to return a completion or error interrupt and overwrites the **rv2, rv3,** and **rv4** words in the structure with data words 1, 2, and 3 returned from the interrupt. The value of the initialization routine is the status code returned with the interrupt. This can be obtained by issuing the CFRSTAT type **ioctl** call previously mentioned.

CFUVRM Updates the VRM. The kernel issues an **Update_VRM** SVC to the VRM. The return value is the return code from the VMI call.

File

/dev/config

Related Information

In this book: "ioctl" on page 2-56, "hft" on page 6-23, "fd" on page 6-17, and "hd" on page 6-20.

The **vrmconfig** command in *AIX Operating System Commands Reference.*

dft, bsc

Purpose

Provides 3270 Distributed Function Terminal (DFT) and Binary Synchronous Communications (BSC) capabilities.

Synopsis

#include <sys/3270.h>

Description

The 3270 AIX device driver is a multiplexed device driver that supports an independent logical 3270 session on each of its channels. It can access multiple VRM device drivers, depending on the special file name used.

Open

The **open** system call initializes a path to a host session. The *oflag* parameter to **open** should be set to **O_RDWR**; all other values are ignored.

The specific adapter and port to be used is specified by the path name that is passed to **open**. The special files named **/dev/dft***n* use a 3278/79 Emulation Adapter. The special files named **/dev/bsc***n* use the Logical Link Control from NETWORK 3270-PLUS (BSC), which uses a Multiprotocol Adapter. To use any **/dev/bsc***n*, you must have NETWORK 3270-PLUS (BSC) or NETWORK RJE-PLUS (BSC) installed on your system. More than one 3278/79 Emulation Adapter or Multiprotocol Adapter can be installed in your system.

A specific session on one of the ports can be addressed by following the special file name with a slash and the device address. For example, `/dev/bsc2/5` identifies device address 5 on a specific port of one of the Multiprotocol Adapters. The path name specified to the **open** system call can specify this device address or not; if it is not specified, then an available device address is used.

The exact correlation between special file names and specific Multiprotocol Adapter ports is established when NETWORK 3270-PLUS (BSC) is installed.

Device address 0 is used to provide profile information for the NETWORK 3270-PLUS (BSC) Logical Link Control and to establish the connection. This applies to BSC only.

Reading and Writing

The *ext* parameter of the **readx** and **writex** extended I/O system calls is used as a pointer to an **io3270** structure that contains additional information. However, some simple applications may not need to supply the *ext* parameter and can use the **read** and **write** system calls.

The **io3270** structure is defined in the **sys/3270.h** header file, and it contains the following members:

```
uint  io_flags;  /* Information Flags */
uint  status;    /* Status Codes      */
```

The value of **io_flags** is formed by logically OR-ing values from the following list:

LOCK Lock or unlock: Lock is set when a **write** or **writex** system call is issued. It is reset (unlocked) when ready for the next write operation.

DATA Data is available to be read.

TRANS Transparent: Indicates that this buffer should be sent in transparent mode, or that it was received in transparent mode. This bit applies only to NETWORK RJE-PLUS (BSC).

COMM Communication check: A communication error was detected.

PROG Program check: A program error was detected.

MACH Machine check: A machine error was detected.

Notes:

1. The application read and write buffers are formatted as 3270 data streams.
2. A **write** or **writex** request fails if the adapter is currently sending outbound data, or if the host issued anything other than a READ BUFFER 3270 command in response to an ATTENTION sent by the VRM device driver.
3. Use the **SND_STATUS ioctl** operation to send status to the host, not **write** or **writex**.

select Support

The 3270 device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are always satisfied immediately.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See "select" on page 2-111 for more information about this system call.

ioctl Operations

The 3270 device driver performs the following **ioctl** operations. See "ioctl" on page 2-56 for a complete description of the **ioctl** system call.

int ioctl (*fildes*, IOCTYPE)
int *fildes*;

Returns a character that identifies the device type. This character is the value given for the **appt** keyword in the **/etc/system** file.

int ioctl (*fildes*, IOCINFO, *arg*)
int *fildes*;
struct devinfo **arg*;

Stores information about the device into the **devinfo** structure, which is defined in the **sys/devinfo.h** header file.

int ioctl (*fildes*, GET_STATUS, *arg*)
int *fildes*;
struct io3270 **arg*;

Gets the device driver status and stores it in the **io3270** structure pointed to by the *arg* parameter. See "Reading and Writing" on page 6-12 for a description of the **io3270** structure.

int ioctl (*fildes*, SND_STATUS, *arg*)
int *fildes*;
struct nsddsstat **arg*;

Sends status and sense data to the host, as specified in the structure pointed to by the *arg* parameter. The **nsddsstat** structure is defined in the **sys/3270.h** header file, and it contains the following members:

```
struct code    co_de
struct status  sta_tus
```

The **code** structure contains the following members:

```
unsigned bit0 : 1;
unsigned bit1 : 1;
unsigned db   : 1;     /* Device Busy */
unsigned us   : 1;
unsigned de   : 1;     /* Device End  */
```

The **status** structure contains the following members:

```
unsigned bit0 : 1;
unsigned bit1 : 1;
unsigned cr   : 1;
unsigned ir   : 1;     /* Intervention Required */
unsigned ec   : 1;
unsigned dc   : 1;
unsigned oc   : 1;
```

Files

/dev/dft0	First 3278/79 Emulation Adapter
/dev/dft1	Second 3278/79 Emulation Adapter
/dev/dft2	Third 3278/79 Emulation Adapter
/dev/dft3	Fourth 3278/79 Emulation Adapter
/dev/bsc0	Identifies a Multiprotocol Adapter port to the BSC LLC
/dev/bsc1	Identifies a Multiprotocol Adapter port to the BSC LLC
/dev/bsc2	Identifies a Multiprotocol Adapter port to the BSC LLC
/dev/bsc3	Identifies a Multiprotocol Adapter port to the BSC LLC.

Related Information

In this book: "ioctl" on page 2-56, "read, readx" on page 2-106, "write, writex" on page 2-184, and "devinfo" on page 4-57.

NETWORK 3270-PLUS (BSC) User Guide.

NETWORK RJE-PLUS (BSC) User Guide.

error

Purpose

Logs system events.

Synopsis

#include <sys/err.h>
#include <sys/erec.h>

Description

The format of an event record depends on the type of event encountered. Each record, however, has a header with the following format:

```
struct errhdr {
    unsigned  e_len;      /* word in record (with header) */
    time_t    e_time;     /* time of day */
    long      e_timex;    /* clock ticks */
    char      e_nid[8];   /* node ID */
    char      e_vmid[8];  /* virtual machine ID */
    union {
       struct  {
          char  ex_class;
          char  ex_subclass[2];
          char  ex_type;  /* record type */
       } ex;
       int csmt;
    } exx;
};
```

The error daemon searches the RAS configuration file **/etc/rasconf** for a stanza labeled **/dev/error**. Minor device 0 of the **error** driver is the interface between a process and the routines that collect error-records in the system. This driver can be opened only for reading by a process (usually the error daemon) with superuser permission. Each read retrieves an entire error record. A read request of less than the entire record causes the retrieved record to be truncated. Multiple processes can open the **error** file to write.

File

/dev/error

Related Information

In this book: "errunix" on page 3-126, "rasconf" on page 4-133, and "errsave" on page C-31.

The **errdemon** command in *AIX Operating System Commands Reference.*

fd

Purpose

Supports the diskette device driver.

Synopsis

#include <sys/devinfo.h>

Description

The diskette special file provides block and character (raw) access to diskettes in the diskette drives, allowing only one process to have a diskette drive open for writing at a time. The **config** device driver associates the minor device number with a particular diskette drive. Normally, the special file **/dev/fd***n* is given the minor device number *n*. Removing the diskette from the drive with diskette files still open may cause various I/O system calls to return errors.

The minor device number specifies both the drive number and the format of the diskette to be read or written. Assume that **/dev/fd***n* corresponds to a diskette drive with minor device number *n*. In this case, **fd0**, **fd1**, **fd2**, and **fd3** specify diskette drives 0 through 3, respectively, without specifying their format.

Using **fs0**, . . . , **fs3**, which correspond to minor device numbers 4 through 7, forces a diskette to be treated as a single-sided diskette. Similarly, **fd0.8**, . . . , **fd3.8**, which correspond to minor device numbers 8 through 11, force the diskette to be treated as an 8-sectored diskette. **fs0.8**, . . . , **fs3.8**, which correspond to minor device numbers 12 through 15, force the diskette to be treated as single-sided and 8-sectored.

Configuration Data

The **config** device driver is called during system initialization to customize diskettes. This is accomplished by calling the device driver at its initialization entry. For diskettes, no device-dependent information is required, so the customize information is the following structure:

```
struct {
    dev_t  devno;              /* major/minor device number */
    unsigned short  iodn;   /* IODN to set */
    unsigned short  ddilen; /* device dependent info length */
    unsigned short  lev;    /* ignored */
```

```
        union  {                    /* optional device dependent info */
               char ddi_fd[];
                 . . .                             /* ddi for other devices */
        } ddi;
    };
```

ioctl Operations

The IOCTYPE type **ioctl** system call returns the device type DD_DISK, defined in the **sys/devinfo.h** header file.

The IOCINFO type **ioctl** system call returns the following structure, defined in the **sys/devinfo.h** header file:

```
    struct devinfo {
        char  devtype;
        char  flags;
        union {
           struct {     /* for disks */
              short  bytpsec;  /* bytes per sector */
              short  secptrk;  /* sectors per track */
              short  trkpcyl;  /* tracks per cylinder */
              long   numblks;  /* number of blocks on diskette */
           } dk;
             . . .                 /* for other devices */
        } un;
    };

    /* flags */
   #define DF_FIXED 01  /* non-removable */
   #define DF_RAND 02   /* random access possible */
   #define DF_FAST 04   /* a relative term */
```

Files

/dev/fd0, /dev/fd1, . . .
/dev/rfd0, /dev/rfd1, . . .

Related Information

In this book: "config" on page 6-7, "fs" on page 4-74, and "ioctl" on page 2-56.

hd

Purpose

Supports the fixed-disk device driver.

Synopsis

#include <sys/devinfo.h>

Description

The fixed-disk device driver provides block and character (raw) access to minidisks on the fixed-disk drives. The **config** device driver associates the minor device number to the minidisk. Normally, the special files **/dev/hd***n* and **/dev/rhd***n* is given the minor device number *n*.

The minidisk with minor device number 0 is always the minidisk used to initially load the system program.

In raw I/O , the buffer must always begin on a full word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise **lseek** system calls should specify a multiple of 512 bytes.

Configuration Data

The **config** device driver is called at its initialization entry point during system initialization to customize minidisks. The following shows the structure of the customize information:

```
struct {
   dev_t  devno;           /* major/minor device number */
   unsigned short iodn;    /* IODN set */
   unsigned short ddilen;  /* device dependent info length */
   short  lev;            /* ignored */
   union  {                 /* optional device dependent info */
      struct {           /* for Send Command SVC */
         int rv2,
         rv3,
         rv4,
         rv5,
              rv6;
              } ddi_sc;
             . . .                              /* ddi for other devices */
          } ddi;
       };
```

If the **iodn** field is 0, the manager receives a **Send_Command** supervisor call (SVC) with values **rv2, rv3**, **rv4, rv5** and **rv6** in registers 2, 3, 4, 5, and 6. The driver waits for the manager to return a completion or error interrupt. The return value is the status code returned at the interrupt, which can be obtained using an **ioctl** system call to the **config** device driver.

ioctl Operations

The VQUERY type **ioctl** call queries the disk write-verify status for the minidisk. It uses the form:

```
ioctl (fildes, command, arg)
struct wverify *arg
```

where **arg** is a pointer to the one-word structure **wverify** that contains the write verify status to be returned. A value of 1 returned indicates enabled and 0 if disabled.

The VCNTRL type **ioctl** call enables or disables write verify for the minidisk. It uses the form:

```
ioctl (fildes, command, arg)
int arg;
```

where an **arg** value of 0 is used to disable disk write-verify and a value of 1 is used to enable disk write-verify.

See "mdverify" on page 3-243 for another way to set and query the write-verify status of a minidisk.

The IOCTYPE type **ioctl** call returns the value DD_DISK, defined in **<sys/devinfo.h>**.

The IOCINFO type **ioctl** call returns the following structure, defined in **<sys/devinfo.h>**:

```
struct devinfo {
   char devtype;
   char flags;
   union {
      struct {        /* for disks */
      short bytpsec;  /* bytes per sector */
      short secptrk;  /* sectors per track */
```

```
       short trkpcyl;  /* tracks per cylinder */
       long numblks;   /* blocks this mini-disk */
      } dk;
         . . .                            /* for other devices */
   } un;
 };

/*flags */
#define DF_FIXED 01  /* non-removable */
#define DF_RAND  02  /* random access possible */
#define DF_FAST  04  /* a relative term */
```

Files

/dev/hd0, /dev/hd1, . . .
/dev/rhd0, /dev/rhd1, . . .

Related Information

In this book: "config" on page 6-7, "fs" on page 4-74, "ioctl" on page 2-56, and "lseek" on page 2-67.

hft

Purpose

Implements a high-function virtual terminal device.

Synopsis

#include <sys/hft.h>

Description

The **hft** device driver supports a virtual terminal concept based on the virtual terminal subsystem of the Virtual Resource Manager. The following information is intended to supplement the discussion of the virtual terminal subsystem in *Virtual Resource Manager Technical Reference.* Additional information can also be found in the file **/usr/lib/samples/README.hft.**

The virtual terminal concept supports the illusion that more devices exist than are physically present and that these devices have characteristics and features not necessarily limited by the actual devices. In addition to displays and keyboards, virtual terminals support locators, valuators, lighted programmable function keys, and sound generators. Virtual terminals are logically independent of each other but share physical resources over time. The virtual terminal that can accept physical input or modify the physical screen at a given time is called the ***active*** virtual terminal.

The virtual terminal provides a model of a single terminal that can be in one of the following modes at a given time:

- Keyboard Send-Receive Mode (KSR)
- Monitor Mode (MOM).

The KSR mode emulates an ASCII terminal using an RT ASCII data stream, which is described in detail in "data stream" on page 5-5. The monitor mode allows applications to have a direct output path to the display hardware and shortened path for keyboard and locator. The form of the data accepted in each mode is unique to that mode. This optimizes the movement of data between the virtual terminal and the application program and supports the different functions within each mode. The default mode is KSR, which supports existing applications expecting an ASCII terminal.

Additional functions supported include:

- Reporting data from input devices such as locators and valuators
- Switching between interactive and noninteractive states
- Changing color palette settings
- Controlling the sound hardware
- Switching between the monitor mode and KSR mode.

The virtual terminal supplies default values for keyboard-to-character mapping, character-to-display mapping, echo/break specification, tab rack, and protocol mode flags to be used until a definition is received from the application.

This **hft** facility is the kernel-level support for virtual terminals. Since the association of virtual terminals to physical terminals is dynamic, this special file, which represents the physical terminal, is multiplexed across virtual terminals by expanding the **open**, **close**, **read**, **write**, and especially the **ioctl** system calls to the driver. This type of driver is specified by the **S_IMPX** bit in the **stat.h** file. Many extra **ioctl** system calls are provided to allow access to advanced features of the **hft** facility. The facilities described in "termio" on page 6-114 also apply to the virtual terminal.

The first (or only) **hft** is minor device 0, and special file **/dev/hft** is associated with it. The special file **/dev/console** is minor device 1. Minor devices 2 and higher are associated with additional **hft** physical terminals, if there are any.

Each time **/dev/hft** is opened, a new **hft** virtual terminal is created and opened. A maximum of 16 virtual terminals can be opened due to limits on system resources.

To reopen an existing virtual terminal, open the special file **/dev/hft/*i***, where *i* is the number of an open driver channel. The channel number can be determined with the HFGCHAN **ioctl** operation. The **/dev/console** special file is channel number 1.

A process can also communicate with the **hft** screen manager by opening the **/dev/hft/mgr** file. Only the screen manager **HFQSMGR** and **HFCSMGR ioctl** operations can be issued to this file. **read** and **write** system calls are not allowed.

The **/usr/lib/samples/hft** directory contains sample programs that use the **hft** virtual terminal subsystem. See the file **/usr/lib/samples/README.hft** for more information about these sample programs.

The following list can be used as a reference to locate where specific topics are discussed:

Contents of hft Section

Initial State

When a new terminal is opened, it is initialized to a known state. This initial state can be changed, if desired. The initial terminal state is the following:

- Mode: Keyboard Send-Receive (KSR).
- Echo/Break Map: Echo all characters; break for none.
- Tab Rack: The first, every eighth, and the last position of every line.
- ASCII Controls:

LNM	Set
IRM	Not set
SRM	Not set
TSM	Not set
CLM	Not set
AUTONL	Set.

- Protocol Mode:

WRAP	Set (wrap cursor at boundary)
HOSTPC	Not set (do not return locator input)
XLATKBD	Set (translate keyboard input)
HOSTS	Not set (do not report keyboard status change)
LPFKS	Not Set (disable lighted programmable function key input)
DIALS	Not set (disable dial or valuator input).

- Locator Threshold: 2.75 millimeters horizontal, 5.5 millimeters vertical.
- Font: Initially, and whenever the physical display device is changed, this is set to be the first font in the customized list of fonts that:

  - Results in a presentation space of 80 columns by 25 rows, and
  - Has a normal appearance (not bold or italic).

  If no font meets these criteria, then the first font that can be displayed on the device is chosen. All alternate fonts are initialized to the selected font.

- Character mode color palette for both foreground and background:

Entry	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

8	Gray
9	Light red
10	Light green
11	Brown
12	Light blue
13	Light magenta
14	Light cyan
15	High intensity white.

termio Support

Input modes described in "termio" on page 6-114 supported are INLCR, IGNCR, ICRNL, IUCLC, IXON, and IXANY. Input modes IGNBRK and BRKINT are not supported because there is no **Break** key. Input modes IGNPAR, PARMRK, and INPCK are not supported because parity is not provided. Input mode ISTRIP is not supported either. ICRNL is supported by using the keyboard remap facility to change the code sent by the **Enter (Return)** and **Ctrl-M** keys. Also, the implementation of IXON is different. If the user presses **Ctrl-S** while output is being performed on the screen, the output does not stop until the end of the current **write** system call.

Output modes supported are OPOST, ONLCR, and OCRNL. The delay insertion, parity, and stop bit modes are not supported.

Line discipline modes supported are ISIG, ICANON, ECHO, ECHOE, ECHOK, ECHONL, NOFLSH, and Enhanced Edit Mode.

Screen paging is also supported using the TCGLEN and TCSLEN **ioctl** operations. When paging is active, the contents of the buffer supplied by the **write** call are written out in page-size pieces.

Other **ioctl** operations supported by **hft** include TCXONC and TCFLSH. The TCSBRK operation is not supported.

select Support

The **hft** device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are always satisfied immediately.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See "select" on page 2-111 for more information about this system call.

ioctl Operations

The **hft** supports a number of operations issued by the **ioctl** system call to provide access to sophisticated features of the **hft**. See "ioctl" on page 2-56 for details about the syntax of the system call itself. For information about issuing requests for these operations to an emulated **hft** device, see "Considerations for hft Emulation" on page 6-54.

Query I/O Error (HFQEIO)

If an I/O operation or other system call to the **hft** fails due to hardware error, the system call returns a nonzero value and sets the **errno** external variable to the value **EIO**. The calling program can get a more detailed device error code by using **ioctl** to issue an HFQEIO operation. This is invoked by the following:

```
int ioctl(fildes, HFQEIO, 0)
int fildes;
```

The return value from the HFQEIO **ioctl** operation is either 0 (indicating that the last I/O operation was successful), -1 (indicating that the HFQEIO operation itself failed), or the error code for the last **hft** I/O operation. See *Virtual Resource Manager Technical Reference* for an explanation of the individual virtual terminal error codes.

Query Device (HFQDEV)

Obtains detailed device information about the types of devices that are associated with the virtual terminal. For details about this query operation, see the Query Device SVC in *Virtual Resource Manager Technical Reference.* This is invoked by the following:

```
int ioctl(fildes, HFQDEV, arg)
int fildes;
struct hfqdev *arg;

struct hfqdev
{
   unsigned short hf_qdrsvd;
   unsigned short hf_qdopts;
   unsigned int   hf_qdlen;
};
```

This **ioctl** operation stores information into an **hfqdresp** structure that overlays the **hfqdev** structure in memory. Only one option is recognized: the **hf_qdopts** field must be set to the value 2. The **hfqdresp** structure contains the following members:

Field	Description
hf_vtrmiodn	The virtual terminal resource manager IODN.
hf_vtrmiocn	The virtual terminal resource manager IOCN.
hf_devtype	The device type. The value in this field is 0x0002 (shared device).
hf_devname[4]	The device name. The value is "VTRM" for "Virtual Terminal Resource Manager."
hf_hwoffset	The offset to hardware characteristics.
hf_devoffset	The offset to the device characteristics.
hf_erroffset	The offset to the error log.

The next eight fields contain information about the last operation that was completed by the virtual terminal for this virtual machine. For more detailed information about these fields, see the discussion of the Query Device SVC in *Virtual Resource Manager Technical Reference.*

Field	Description
hf_newics	New Interrupt Control Status register value.
hf_statflags	Status flags.
hf_ovrncnt	Overrun count.
hf_opresult	Operation result.
hf_deviodn	Device IODN.
hf_dataword1	Device-dependent data or command extension segment ID.
hf_dataword2	Device-dependent data or command extension address.
hf_dataword3	Device-dependent data.
hf_ddilen	The length (in words) of the device-dependent information.
hf_rc	The return code from IPL or the **Define_Device** SVC. A value of 0 indicates a successful operation.
hf_smiocn	The screen manager IOCN.
hf_vtmpiocn	The virtual terminal mode processor IOCN.
hf_keyiodn	The keyboard IODN.
hf_lociodn	The locator IODN. This field is set at IPL time. Note that the locator is optional. If the locator is not used, this field is be set to 0.
hf_spkiodn	The speaker IODN. This field is set at IPL time.

hf_numfont The number of fonts. One font is supplied by the VRM; however up to 31 additional fonts can be defined.

hf_fontiocn[*i*] The font IOCNs. Undefined font IOCN fields are set to 0.

hf_numdisp The number of physical displays. This field is completed at IPL time. The VRM supports as many as 4 physical displays.

The next three fields (**hf_devid, hf_deviodn,** and **hf_deviocn**) are repeated four times to accommodate additional displays. Fields in this array that are not used are set to 0.

hf_physd[*i*]**.hf_devid**
Contains a value that is a code identifier for a particular physical device, such as a display adapter or monitor combination, in use.

hf_physd[*i*]**.hf_deviodn**
The IODN for the physical display device.

hf_physd[*i*]**.hf_deviocn**
The IOCN for the virtual display driver.

hf_keymapiocn The IOCN that defines how key positions map to characters.

hf_chrmapiocn The IOCN that defines how characters map to display codes for the Unique 1 and Unique 2 character sets.

hf_echomapiocn The IOCN that defines the echo and break maps.

hf_initiocn The IOCN of the virtual terminal mode processor initialization parameters. Examples of these parameters include protocol modes, tab rack, and so on.

hf_dialsiodn The IODN of the valuator dial device driver.

hf_lpfkiodn The IODN of the lighted program function key device driver.

Reconfigure (HFRCONF)

A user program can reconfigure the virtual terminal to include different real devices. Helpful information about virtual terminal reconfiguration can be found in the file **/usr/lib/samples/README.conf**. This operation is invoked by the following:

```
int ioctl(fildes, HFRCONF, arg)
int fildes;
struct hfrconf *arg;

struct hfrconf
{
   unsigned hf_op;
   unsigned hf_obj;
```

```
        union
        {
          uint hf_infob;
          struct
          {
             ushort hf_iodn;
             ushort hf_iocn;
          } hf_info2;
        }hf_info;
      };
```

This command changes the configuration of the physical terminal or the virtual terminal defaults. For example, there can be up to four display devices, one locator, one speaker, and up to 32 fonts associated with a real terminal.

The **hf_op** field contains the requested operation. The valid operations appear in the following list. These reconfigure operations, with the exception of those followed by an * (asterisk), take effect only for terminals opened after the reconfiguration. The operations followed by an asterisk take effect for the terminals that are currently open as well as those opened after the reconfiguration.

HFADDLOC Adds a real locator. **hf_obj** contains the real locator device driver IODN.

HFADDSOUND Adds a real sound device. **hf_obj** contains the real sound device driver IODN.

HFADDDISPLAY Adds a real display. The following fields must also be set for this operation:

hf_obj	The real display identifier
hf_iodn	The display device driver IODN
hf_iocn	The virtual display driver IOCN.

HFDELDISPLAY Deletes a real display. **hf_obj** contains the real display identifier.

HFADDFONT Adds a font. **hf_obj** contains the font IOCN.

HFCHGKBDRATE* Changes the keyboard typematic rate. Bits 24 − 31 of **hf_obj** indicate the keyboard typematic rate. For the standard RT PC keyboard, valid values are between 2 and 40 characters per second and can be incremented in 1 character-per-second units. The default value for the RT PC keyboard is 14 characters per second.

HFCHGKBDDEL* Changes the keyboard typematic delay. Bits 16 − 31 of **hf_obj** indicate the keyboard typematic delay. For the standard RT PC keyboard, valid values are between 300 and 600 milliseconds and can be incremented in 100 millisecond units. The default value for the RT PC keyboard is 400 milliseconds.

HFCHGLOCRATE* Change locator sample rate. Bits 24 − 31 of **hf_obj** indicate the locator sample rate. For the standard RT PC locator, valid values are 10, 20, 40, 60, 80, and 100 samples per second. The default for the RT PC locator is 60 samples per second.

HFCHGCLICK* Turns the keyboard click mechanism on or off. Bit 31 of **hf_obj** indicates whether the speaker produces a sound when a key is pressed. Sound is suppressed when bit 31 equals 0 and produced when bit 31 equals 1. The default for the RT PC keyboard is keyboard click on.

HFCHGVOLUME* Sets the sound volume level. Bits 24 − 31 indicate the volume of sounds produced by the speaker. For the standard RT PC speaker, valid values are 0 (sound off) and, 1 (low volume), 2 (medium volume) and 3 (high volume). The default for the RT PC speaker is medium volume.

HFKEYMAP Replaces the position code map. **hf_obj** contains the new position code map IOCN. See the **/usr/lib/samples/hft/hftkbdmap.c** file.

HFDISPMAP Replaces the character code maps for the Unique 1 and Unique 2 character sets. See the **/usr/lib/samples/hft/hftchrmap.c** file. **hf_obj** contains the new unique character code map IOCN.

HFECHOMAP Replaces the echo/break map. **hf_obj** contains the new echo/break map IOCN. See the **/usr/lib/samples/hft/hftecbrmap.c** file.

HFDEFAULT Replaces miscellaneous default values. **hf_obj** contains the new miscellaneous defaults IOCN. See the **/usr/lib/samples/hft/hftmiscdef.c** file.

HFSETDD Changes the default display. **hf_obj** contains the real display identifier.

HFADDDIALS Adds a real dial device. **hf_obj** contains the dial device driver IODN.

HFADDLPFK Adds a real lighted programmable function key (LPFK) device. **hf_obj** contains the LPFK device driver IODN.

HFCHNGDMA Changes the DMA start address and length. **hf_obj** contains the new DMA start address, and **hf_infob** contains the length of the new DMA area.

Get Channel Number (HFGCHAN)

Returns the current driver channel number as the value of the **ioctl** system call. This number can be used to open a specific virtual terminal. The *arg* parameter is ignored. This is invoked by the following:

```
int ioctl (fildes, HFGCHAN, 0)
int fildes;
```

Set Echo and Break Maps (HFSECHO)

Sets the **hft** echo and break maps. ***Echoing*** displays the character associated with a keystroke on the screen or performs the function associated with a control. ***Breaking*** switches the input path from the monitor mode input buffer to the unsolicited ASCII datastream flow. Echoing applies only to KSR mode; breaking applies only to MOM mode. Echoing and breaking can be selectively enabled for each ASCII code point and multi-byte control sequence. The default is to echo all characters and control sequences, but not to break on any of them.

The HFSECHO operation is invoked by the following **ioctl** call:

```
int ioctl(fildes, HFSECHO, arg)
int fildes;
struct hfbuf *arg;

struct hfbuf
{
    char *hf_bufp;
    int hf_buflen;
};
```

The **hf_bufp** field points to an array of 32 integers. The **hf_buflen** field contains the value 128 (0x80), which is the length of the array in bytes. The first sixteen integers constitute the echo map; the second sixteen integers are the break map.

Each of the two maps is treated as a set of bits. Bit 0 is the most significant bit of the first integer. Bit 511 is the least significant bit of the sixteenth integer. Each bit corresponds to an ASCII code point or multi-byte control. Bits 0 through 255 (0xFF) correspond to the single-byte codes. Bits 256 (0x100) and higher correspond to multi-byte control sequences, as illustrated in Figure 6-1 on page 6-35. Bit 511 (0x1FF) specifies whether to echo or break on invalid and unsupported multi-byte control sequences. See "data stream" on page 5-5 for a detailed explanation of each of the multi-byte control sequences.

|< — Most Significant Hex Digits 0,1 — >|

Hex2	10		12	13	14	. . .		1F	
0	CBT	DMI		RC					
1	CHA	EMI	RI	KSI					
2	CHT	EA		VTD					
3	CTC	ED	RIS						
4	CNL	EF	RM						
5	CPL	EL	SD						
6	CPR	ECH	SL						
7	CUB	GSM	SR						
8	CUD	HTS	SU						
9	CUF	HVP	SGR						
A	CUP	ICH	SG0						
B	CUU	IL	SG1						
C	CVT	IND	SM						
D	DCH	NEL	TBC						
E	DL	PFK	VTS						
F	DSR		SC					INV	

Figure 6-1. Bit Positions of ASCII Controls in Echo Map

For the echo map, a bit set to 1 means the character or control sequence is echoed when a key that is mapped to it is pressed. The echo map is active only in KSR mode and can be set only from KSR mode.

For the break map, a bit set to 1 means that the character or control sequence is reported using the **read** system call instead of being placed in the input ring buffer. Also, the **SIGMSG** signal is sent to the process to indicate that input data is available. The break map is active only in monitor mode. (See "Monitor Mode (MOM)" on page 6-73 for a description of the input ring buffer.)

The echo and break maps are shared by all code pages. For P0 graphic code points (0x20 to 0xFF), bits 32 to 255 (0x20 to 0xFF) of each map are used. For other code pages, each half of the code page is associated with bits 128 to 255 (0x80 to 0xFF). For example, bit 160 (0xA0) specifies the echo or break status of code points P0 0xA0, P1 0x20, P1 0xA0, P2 0x20, and P2 0xA0.

Set Keyboard Map (HFSKBD)

Sets the keyboard map. Most keys on the keyboard can be remapped, changing the character or control sequence each key generates when pressed. See "keyboard" on page 6-78 for additional details. This is invoked by the following:

```
int ioctl(fildes, HFSKBD, arg)
int fildes;
struct hfbuf *arg;

struct hfbuf
{
   char *hf_bufp;
   int   hf_buflen;
};
```

The **hf_bufp** field points to a **hfkeymap** structure, and **hf_buflen** contains its length.

```
struct hfkeymap {
        char  hf_rsvdl ;
        char  hf_nkeys;
        struct hfkey {
                char hf_kpos;
                char hf_kstate;
                struct hfkeyasgn
                {
                     /* for single character */
                     char hf_pagenum;    /* Code page  */
                     char hf_character;   /* Character to map  */
#define nf_page      hf_pagenum
#define hf_char      hf_character
                     /* for function id */
#define hf_keyidh    hf_pagenum       /* high byte of id */
#define hf_keyidl    hf_character     /* low byte of id */
                     /* for character string */
#define hf_kstrl     hf_character     /* length of string */
                }hf_keyasn;
        } hfkey[HFNKEYS];
};
```

The **hfkeymap** structure can remap one or more keys, the number of which is specified by the **hf_nkeys** field. This many **hfkey** structures follow. **HFNKEYS**, which is used as the dimension for the **hfkey** array, is by default defined to be 1, allowing one key to be remapped. To change **HFNKEYS**, set its value in a **#define** statement that comes before the **#include <hft.h>** statement.

The **hfkey** structure contains information for each key being remapped, such as key position, shift states, and the type of remapping being done. The fields in the **hfkey** structure are:

hf_kpos The key position number. See “keyboard” on page 6-78.

hf_kstate This field is subdivided into three groups of bits:

HFMAPMASK
Defines the bits that specify the type of mapping to be performed:

HFMAPCHAR	Specifies mapping a single character to a key.
HFMAPNONSP	Specifies mapping a nonspacing character to a key. (See “data stream” on page 5-5 for informxtion about nonspacing characters.)
HFMAPFUNC	Specifies mapping a function ID to a key.
HFMAPSTR	Specifies mapping a string of more than one character to a key.

HFSHFMASK
Defines the bits that specify the shift state that applies to the key being mapped:

HFSHFNONE	Specifies the base state (no shift state)
HFSHFSHFT	Specifies the shift state
HFSHFCTRL	Specifies the **Ctrl** state
HFSHFALT	Specifies the **Alt** state
HFSHFALTGR	Specifies the **Alt Gr** (Alternate Graphics) state.

HFCAPSL
Specifies whether the **Caps Lock** state affects the key. If set, then when **Caps Lock** mode is on, the base state of a key functions as the shift state, and the shift states functions as the base state.

The **hfkeyasgn** structure specifies the key to be remapped and the character codes generated when the key is pressed or released. The fields of this structure differ depending on the value of the **HFMAPMASK** bits in **hf_kstate**:

HFMAPCHAR, HFMAPNONSP:

hf_page	Specifies the code page
hf_char	Specifies a character (also called a code point) in that code page.

HFMAPSTR:

hf_page	Specifies the code page
hf_kstrl	Specifies (*the length of the string in bytes*) minus 1.

This is immediately followed by the string.

Note: Due to limitations of the **hfkeymap** structure, only one key can be assigned a string value, and it must be the last key specified in the **hfkey** array. This is because the structure itself does not contain space for the variable-length string, but the string must immediately follow the structure in memory. The virtual terminal subsystem supported by the VRM allows any number of keys to be assigned string values, and you can you can do so if you set up your own key map buffer instead of using **hfkeymap**.

HFMAPFUNC:

hf_keyidh Specifies the high-order byte of the function ID.
hf_keyidl Specifies the low-order byte of the function ID.

The following list gives the function IDs for each of the functions that can be assigned to keys. See "Multi-Byte Controls" on page 5-13 for more details about these functions.

ID **Name**

0x0000 — 0x00FE
(PFK) Issues the Programmable Function Key sequence for PF key 1 (ID = 0x0000) through 255 (ID = 0x00FE).
0x0101 (CUU) Moves the application cursor up one line.
0x0102 (CUD) Moves the application cursor down one line.
0x0103 (CUF) Moves the application cursor forward one character.
0x0104 (CUB) Moves the application cursor backward one character.
0x0105 (CBT) Moves the application cursor to the previous horizontal tab stop or beginning of field.
0x0106 (CHT) Moves the application cursor to the next horizontal tab stop or beginning of field.
0x0107 (CVT) Moves the application cursor down one vertical tab stop.
0x0108 (HOME) Moves the application cursor to the first line, first character in the presentation space.
0x0109 (LL) Moves the application cursor to the last line, first character in the presentation space.
0x010A (END) Moves the application cursor to the last line, last character in the presentation space.
0x010B (CPL) Moves the application cursor to the first character of the previous line.
0x010C (CNL) Moves the application cursor to the first character of the next line.
0x0151 (DCH) Deletes the character over the application cursor.
0x0152 (IL) Inserts one line following the line of the application cursor.
0x0153 (DL) Deletes the line of the application cursor.
0x0154 (EEOL) Erases to the end of the line.
0x0155 (EEOF) Erases to the next tab stop.
0x0156 (CLEAR) Erases all characters from the presentation space.
0x0157 (INIT) Restores the initial state of the virtual terminal. (See the description of RIS in "Multi-Byte Controls" on page 5-13.)

0x0162	(RI) Performs one line reverse index control.
0x0163	(IND) Performs one line index control.
0x1FFF	(IGNORE) Sends no information when the key is pressed.

Note:

On the U.S. 101-key keyboard, the left **Alt** key produces the **Alt** shift state, and the right **Alt** key produces the **Alt Gr** shift state. The default keyboard mapping for the **Alt** and **Alt Gr** states is identical for all keys.

If a U.S. 101-key keyboard is attached, then mapping the **Alt** state of a key automatically causes the same mapping to be assigned to the **Alt Gr** state. This allows the two **Alt** keys on the U.S. keyboard to function identically for most applications. If you want to remap both the **Alt** and **Alt Gr** states of a key, you must remap the **Alt** state first, then the **Alt Gr** state. Software written primarily for keyboards other than the U.S. keyboard should remap the states in this order to assure compatibility.

If the Japanese 106-key keyboard is attached, then access to the **Alt Gr** shift state is not possible.

Get Virtual Terminal ID (HFGETID)

Gets identification information for the current **hft** virtual terminal. This is invoked by the following:

```
int ioctl(fildes, HFGETID, arg)
int fildes;
struct hfgetid *arg;

struct hfgetid  {
    unsigned hf_iodn;
    unsigned hf_pgrp;
    unsigned hf_chan;
};
```

The **hf_iodn** field is the I/O device number of the virtual terminal. The **hf_pgrp** field is the process group ID; that is, the process ID of the terminal group leader. The **hf_chan** field is the channel number that is also returned by the HFGCHAN **ioctl** operation.

Query (HFQUERY)

This gets information about the current virtual terminal. This is invoked by the following:

```
int ioctl(fildes, HFQUERY, arg)
int fildes;
struct hfquery *arg;
struct hfquery  {
    char *hf_cmd;
    int hf_cmdlen;
    char *hf_resp;
    int hf_resplen;
};
```

The first two fields describe a buffer containing the command. The second two fields describe a buffer large enough to hold the expected response. Note that each command and response structure begins with a ***virtual terminal data*** (VTD) header. (See "Output" on page 6-61 for an explanation of the VTD header.) The following query commands use this **ioctl** operation.

Query Device IDs Command

This command uses the **hfqdevidc** structure, which contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFQDEVIDCH**
hf_intro.hf_typelo	**HFQDEVIDCL**

This command fills the response buffer with the information about the display devices. The information is returned in an **hfqdevidr** structure, which has the following fields:

Field	Value
hf_intro.hf_typehi	**HFQDEVIDRH**
hf_intro.hf_typelo	**HFQDEVIDRL**
hf_numdev	The number of devices for which data is reported.

The following fields are repeated for each physical device:

hf_devid Physical device ID.

The first device ID is the active display device ID, unless the **change physical display** command has changed the active display ID. The following values are possible:

0x0401*mmnn*	IBM PC Monochrome Adapter and PC Monochrome Display (5151)
0x0402*mmnn*	Advanced Monochrome Graphics Adapter and Display (6153)
0x0403*mmnn*	Enhanced Graphics Adapter and PC Monochrome Display (5151)
0x0404*mmnn*	Enhanced Graphics Adapter and Display (5154)

0x0405*mmnn*	Extended Monochrome Graphics Adapter and Display (6155)
0x0406*mmnn*	Advanced Color Graphics Adapter and Display (6154)
0x0408*mmnn*	IBM 5081 Graphics Adapter and Display.

Note: The *mm* value indicates whether the adapter is totally functional. When this value is 0x00, the adapter is totally functional. Any other value indicates the adapter is less than fully functional or not working at all, but is present on the machine. The *nn* value can be from 0x01 to 0x04 and differentiates between multiple instances of the same adapter type.

hf_class Display class (0x44).

Query Physical Device Command

This command returns information about display or locator devices. The **hfqphdevc** structure is used to issue this command:

Field	Value
hf_intro.hf_typehi	**HFQPDEVCH**
hf_intro.hf_typelo	**HFQPDEVCL**
hf_phdevid	Physical device ID. The value 0 specifies the active device that is currently attached to the virtual terminal.

The response to this command gives the following information:

```
struct hfqphdevr
{
   char hf_intro[HFINTROSZ];
   char hf_sublen;
   char hf_subtype;
   /* locator  device */
   char hf_scale[4];
   char hf_locattr[1];
   char hf_rsvd[3];
   /* display device */
   char hf_attrib[4];
   char hf_pwidth[4];
   char hf_pheight[4];
   char hf_mwidth[4];
   char hf_mheight[4];
   char hf_bperpel[4];
   char hf_phdevid[4];
   /* display font */
   char hf_numfont[4];
   /* remainder is of variable length */
   /* struct hffont hffont[N];  where N is value in hf_numfont */
   char hf_fontstart;
    /* following is one color response */
    /* struct hfcolor hfcolor; */
 };

 struct hfqfont
 {
    char hf_fontid[4];
    char hf_fontstyle[4];
    char hf_fontattr[4];
    char hf_fontwidth[4];
    char hf_fontheight[4];
};

struct hfcolor
{
   char hf_numcolor[4];
   char hf_numactive[4];
   char hf_numfgrnd[4];
   char hf_numbgrnd[4];
   char hf_actcolor[4];
};
```

These structures are explained in the following sections that have headings beginning with the word *Physical.*

Physical Device Information VTD Header

Field	**Value**
hf_intro.hf_typehi	**HFQPDEVRH**
hf_intro.hf_typelo	**HFQPDEVRL**

Physical Locator Information

Field	**Value**
hf_scale	Scale factor (millimeters per 100 counts)
hf_locattr[0]	Locator attributes: **HFLOCABS** If set, then the locator device reports absolute coordinates (for example, a tablet device). If not set, then it reports relative coordinates (for example, a mouse).

Physical Display Device Information

Field	**Value**
hf_attrib[0]	Display device attributes: **HFISAPA** All-points-addressable (APA) display. **HFHASBLINK** Blink function allowed. All other values are reserved.
hf_attrib[2]	Display device attributes: **HFHACOLOR** Color allowed. All other values are reserved.
hf_attrib[3]	Display device attributes: **HFCHGPALET** Can change display adapter's color palette. All other values are reserved.
hf_pwidth	Displayable width of physical screen, expressed in ***picture elements*** (also called pels or pixels) for all displays.
hf_pheight	Displayable height of physical screen, expressed in pels for all displays.
hf_mwidth	Displayable width (in millimeters).
hf_mheight	Displayable height (in millimeters).
hf_bperpel	Bits per pel (1, 2 or 4).
hf_phdevid	Display device ID.

Physical Display Font Information

Field	Value
hf_numfont	Number of fonts available to this display. The following fields appear for each available font.
hf_fontid	Physical font ID.
hf_fontstyle	Physical font style.
hf_fontattr[0]	Physical font attribute. This field may have the following values: **HFFNTPLAIN** Plain **HFFNTBOLD** Bold **HFFNTITALIC** Italic.
hf_fontwidth	Physical font width (the width of a character cell in pels).
hf_fontheight	Physical font height (the height of a character cell in pels).

Physical Display Color Information

Field	Value
hf_numcolor	Total number of colors possible
hf_numactive	Number of colors that can be active at any one time
hf_numfgrnd	Number of foreground color options
hf_numbgrnd	Number of background color options
hf_actcolor	Active color value. The value of this field can be in the range 0 to the total number of colors possible (**hf_numcolor**) minus 1. This field is repeated for each of the currently active colors.

Query Locator Command

To query the locator, use the **hfqgraphdev** structure with fields set as follows:

Field	Value
hf_intro.hf_typehi	**HFQLOCCH**
hf_intro.hf_typelo	**HFQLOCCL**

This command returns a **hfqlocr** structure with the following fields:

Field	Value
hf_intro.hf_typehi	**HFQLOCRH**

hf_intro.hf_typelo	**HFQLOCRL**	
hf_resolution	The resolution of the locator (a 4-byte value).	
hf_devinfo[0]	Locator attributes:	
	HFLOCABS	If set, absolute coordinates (tablet). If not set, relative coordinates (mouse).
	HFLOCUNKNOWN	Unknown sensor type, or the locator is a mouse.
	HFLOCSTYLUS	The tablet has a stylus sensor.
	HFLOCPUCK	The tablet has a puck sensor.
hf_horzmax_cnt	Horizontal maximum count (a 2-byte value).	
hf_vertmax_cnt	Vertical maximum count (a 2-byte value).	
hf_horzdead_zone	Horizontal tablet dead zone or mouse threshold.	
hf_vertdead_zone	Vertical tablet dead zone or mouse threshold.	

Query LPFKs Command

To query the LPFKs, use the **hfqgraphdev** structure with fields set as follows:

Field	**Value**
hf_intro.hf_typehi	**HFQLPFKSCH**
hf_intro.hf_typelo	**HFQLPFKSCL**

This command returns a **hfdial_lpfk** structure with the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFQLPFKSRH**
hf_intro.hf_typelo	**HFQLPFKSRL**
hf_numlpfks	Number of LPFKs on the device.
hf_data2.lpfk.flags	A set of 32 bits corresponding to each of the LPFKs. Bits that are set to 1 indicate enabled LPFKs; bits set to 0 indicate disabled LPFKs.

Query Dials Command

To query the dials, use the **hfqgraphdev** structure with fields set as follows:

Field	Value
hf_intro.hf_typehi	**HFQDIALSCH**
hf_intro.hf_typelo	**HFQDIALSCL**

This command returns a **hfdial_lpfk** structure with the following fields:

Field	Value
hf_intro.hf_typehi	**HFQDIALSRH**
hf_intro.hf_typelo	**HFQDIALSRL**
hf_numdials	Number of dials on the device.
hf_data2.granularity	An array of sixteen 1-byte values giving the granularity of each dial. Granularity is the number of events per full 360° revolution of the dial. The values in the array represent powers of 2.

Query Presentation Space Command

This data determines how to define a block of characters in the presentation space to query. Attribute and character set information on the queried block are returned. This query is valid only in KSR mode. (Note that this operation is called "Query ASCII Codes and Attributes" in *Virtual Resource Manager Technical Reference.*)

The **hfqpresc** structure is used for this command, and it contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFQPRESCH**
hf_intro.hf_typelo	**HFQPRESCL**
hf_sublen	2
hf_subtype	0
hf_xuleft	The upper-left X coordinate (first column of the block)
hf_yuleft	The upper-left Y coordinate (first row in the block)
hf_xlright	The lower-right X coordinate (last column in the block)
hf_ylright	The lower-right Y coordinate (last row in the block).

The data returned from this command is an ASCII data stream that contains character codes from the queried block. Character set and attribute changes are indicated with SGR and SG0 control sequences. A line feed control is returned after the last character code in each line of the queried block.

Note: The returned attributes may be only a subset of the original attributes specified for query. The subset in this case is those attributes actually supported by the physical device.

The response is returned in an **hfqpresr** structure, which contains the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFQPRESRH**
hf_intro.hf_typelo	**HFQPRESRL**

The response contains an ASCII data stream that includes all ASCII data currently associated with the input buffer.

Query HFT Device Command

This command gets information about the **hft** device. To issue this command, use the **hfqhftc** structure with fields set as follows:

Field	**Value**
hf_intro.hf_typehi	**HFQHFTCH**
hf_intro.hf_typelo	**HFQHFTCL**

The command returns an **hfqhftr** structure with the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFQHFTRH**
hf_intro.hf_typelo	**HFQHFTRL**
hf_phdevid	Physical display device ID (the same as returned by "Query Device IDs Command" on page 6-40)
hf_phrow	Number of character rows, based on the current font
hf_phcol	Number of character columns, based on the current font
hf_phcolor	Number of colors allowed on the display
hf_phfont	Number of fonts defined in the system
hf_phkbdid	Physical keyboard ID: 0 101-key keyboard 1 102-key keyboard 2 106-key keyboard.

Query DMA Command

This command queries the starting address and length of the application's DMA area. To issue this command, use the **hfqdmac** structure with fields set as follows:

Field	**Value**
hf_intro.hf_typehi	**HFQDMACH**
hf_intro.hf_typelo	**HFQDMACL**

The command returns an **hfqdmar** structure with the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFQDMARH**
hf_intro.hf_typelo	**HFQDMARL**
hf_dmaaddr	Starting address of the DMA area
hf_dmalen	Length of the DMA area.

Enable Sound Signal (HFESOUND)

This command informs the terminal driver of the intent to use sound, enabling the routing of the sound response signal. This is invoked by the following:

```
int ioctl(fildes, HFESOUND, arg)
int fildes;
struct hfsmon *arg;

struct hfsmon
{
  int hf_momflags;
};
```

The **hf_momflags** field contains one of the following values:

HFSINGLE Only the process issuing the **ioctl** system call is to receive a sound response signal.

HFGROUP All members of the current process group are to receive a sound response signal.

Disable Sound Signal (HFDSOUND)

This informs the terminal driver of the intent to discontinue the use of sound. Sound response signals are not sent. This is invoked by the following:

```
int ioctl (fildes, HFDSOUND, 0)
int fildes;
```

Enter Monitor Mode (HFSMON)

This requests monitor mode. Monitor mode provides a program with direct control of the screen and keyboard. This is invoked by the following:

```
int ioctl(fildes, HFSMON, arg)
int fildes;
struct hfsmon *arg;

struct hfsmon
{
  int hf_momflags;
};
```

The **hf_momflags** field contains one of the following values:

HFSINGLE Only the process issuing the **ioctl** system call is to receive monitor mode signals.

HFGROUP All members of the current process group are to receive monitor mode signals.

Exit Monitor Mode (HFCMON)

Releases monitor mode. This is invoked by the following:

```
int ioctl(fildes, HFCMON, 0)
int fildes;
```

Query Screen Manager (HFQSMGR)

Queries the screen manager. The file descriptor must be associated with a screen manager, that is, **/dev/hft/mgr**. This is invoked by the following:

```
int ioctl(fildes, HFQSMGR, arg)
int fildes;
struct hfbuf *arg;
struct hfbuf
{
   char *hf_bufp;
   int hf_buflen;
};
```

The contents of the following **hfqstat** structure are stored in the memory area pointed to by **hf_bufp**.

```
struct hfqstat
{
   short hf_numvts;
   struct hfvtinfo
   {
     unsigned short hf_vtiodn;
     unsigned short hf_vtstate;
   } hf_vtinfo[HFNUMVTS];
};
```

Field	**Description**
hf_numvts	The number of virtual terminals.

The following fields are repeated for each virtual terminal:

hf_vtiodn The virtual terminal IODN.

hf_vtstate Status:

HFVTHIDDEN	The virtual terminal is hidden.
HFVTACTIVE	The virtual terminal is active.
HFVTCOMMAND	The virtual terminal is the command terminal.

Control Screen Manager (HFCSMGR)

This commands the screen manager. This command controls the status of virtual terminals. Virtual terminals are linked together in a group called the ***screen manager ring***. The screen manager places an entry in the ring for each virtual terminal opened. The terminal that is currently active is called the ***head*** of the ring; the last terminal on the ring is called the ***tail***. When a new terminal is added to the ring, the terminal becomes the head of the ring.

Two key sequences switching between virtual terminals and control which terminal is currently active. The ***active*** terminal is the terminal that accepts keyboard or locator input and updates the physical display. Pressing the **Alt + Action** keys on the active terminal makes the ***next*** virtual terminal active. This relationship is indicated by **a** in

Figure 6-2 on page 6-51. Pressing the **Shift + Action** on the active terminal makes the ***last*** virtual terminal active. The **b** in Figure 6-2 on page 6-51 indicates this relationship.

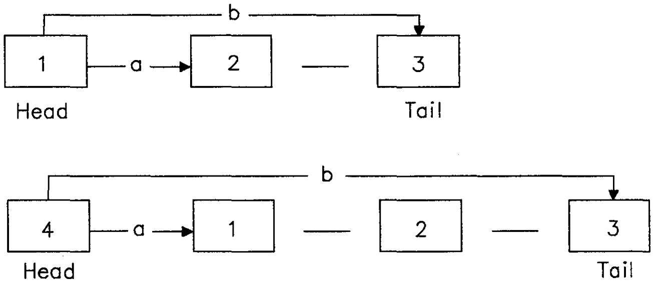


Figure 6-2. Screen Manager Ring Examples. In this figure, **a** indicates the path from the active virtual terminal to the next and **b** indicates the path from the active virtual terminal to the last.

Note that with three entries in the ring, all the terminals can be accessed with a single key sequence. With four or more entries, terminals can be skipped in some cases to activate a particular terminal. For example, in the preceding figure with four terminal entries, terminal #2 cannot be accessed from the active terminal #4 without first skipping to terminal #1 or terminal #3.

The ***hide*** option of this command logically removes terminals from the ring. Hiding a terminal causes it to be bypassed when its position in the ring ordinarily makes it the active terminal.

The file descriptor must be associated with a screen manager, that is, **/dev/hft/mgr**. This is invoked by the following:

```
int ioctl(fildes, HFCSMGR, arg)
int fildes;
struct hfsmgrcmd *arg;

struct hfsmgrcmd {
    int hf_cmd;
    int hf_vtid;
    int hf_vsid;
};
```

The **hf_vtid** and **hf_vsid** fields are set as follows:

hf_vtid The IODN of the virtual terminal

hf_vsid 0.

The **hf_cmd** field contains one of the following screen manager commands:

SMACT Activates the virtual terminal. This command places the virtual terminal specified by the IODN at the head of the screen manager ring, making it the active terminal. The terminal's hidden flag is also cleared. The screen manager cannot activate the virtual terminal if the currently active virtual terminal cannot be deactivated.

SMHIDE Hides the virtual terminal. This command marks the terminal identified by the IODN so that the screen manager will not activate it. This does not affect the terminal's position in the ring. When the hidden flag is set, the screen manager ignores the terminal's presence in the ring until an **SMUNHIDE** command is issued. If the virtual terminal is active when the hide command is issued, then the screen manager makes the terminal inactive (if possible), but does not prevent the virtual machine attached to it from communicating with it. Hiding the active virtual terminal has the same effect as the **last** window function. If all virtual terminals are hidden, then the physical display continues to show the contents of the last virtual terminal that was hidden.

SMSCMD Sets the command virtual terminal. This command designates a terminal as the command virtual terminal. The ***command virtual terminal*** is the terminal that is activated by pressing both locator buttons at the same time, or by pressing the **Ctrl-Action** key sequence.

SMUNHIDE Undoes the action performed by **SMHIDE**. The **hf_vtid** field contains the IODN of the virtual terminal where the command should be sent. The **hf_vsid** field is reserved.

This command restores the presence of the terminal in the ring, but does not affect its ring position or make it active. If the virtual terminal happens to be at the head of the ring when this command is issued, then it becomes visible and active.

SMCVTEN Causes the command virtual terminal to be activated when both locator buttons are pressed at the same time. This is the default setting. Since all virtual terminals are affected, programs that change this setting should restore it as soon as the locator is no longer needed.

SMCVTDI Causes input data to be reported when both locator buttons are pressed at the same time. The data reported is similar to that reported when a single button is pressed.

DMA Move (HFMDMA)

Specifies the source, destination, and length of a Monitor Mode DMA move data operation. This is invoked by the following:

```
int ioctl(fildes, HFMDMA, arg)
int fildes;
struct hfmdma *arg;

struct hfmdma
{
   uint hf_srcaddr;   /* Virtual source address of DMA data      */
   uint hf_dmalen;    /* Length of data to be moved by DMA       */
   uint hf_destaddr;  /* Virtual destination address of DMA data */
}
```

This **ioctl** operation maps an application data area to segment E DMA space or unmaps an application data area. Mapping occurs when the source address specifies the application data area and the destination address specifies the segment E space. Unmapping takes place when the addresses are reversed from mapping. Pages are pinned when mapping takes place and unpinned during unmapping.

The range of space available for segment E DMA depends on the amount of memory installed in the system and on the amount of memory allocated to DMA space by the **HFCHNGDMA** option of the **HFRCONF ioctl** operation (see "Reconfigure (HFRCONF)" on page 6-31).

Warning: Once a data space is mapped onto segment E space with this **ioctl** operation, do not attempt to access the data space until after it is unmapped. Otherwise, loss of data, unpredictable results, and permanent depletion of system resources may occur.

Considerations for hft Emulation

Communicating with an emulated or remote **hft** device presents a unique situation because the **ioctl** system call cannot be used. This is a result of the fact that **ioctl** passes data directly to the virtual terminal subsystem, bypassing the data stream. An **hft** emulator is usually connected through a pseudo-tty device, which means that all communication with it must be done through the data stream. Pseudo-tty devices are discussed under "pty" on page 6-107.

Therefore, two special multi-byte control sequences can be used in place of invoking the **ioctl** system call, allowing applications to request an emulated **hft** to perform the **ioctl** operations. However, the **hft** device driver, which controls the local console, does not recognize these control sequences; you must still use **ioctl** to perform these operations on an **hft** device that is not emulated.

Both of these multi-byte control sequences begin with a virtual terminal data (VTD) header. VTDs are explained under "Output" on page 6-61.

To perform an **hft ioctl** operation whether or not the **hft** is emulated, an application should do the following:

1. Determine whether the **hft** device is being emulated. If the call **ioctl** **(***fildes***, IOCTYPE, 0)** returns the value **DD_PSEU**, then *fildes* is a pseudo-tty device, which means that *fildes* may be connected to an **hft** emulator. Otherwise, the **hft** device is not emulated.
2. If the **hft** is not emulated, then issue a regular **ioctl** system call, as outlined in "ioctl Operations" on page 6-29.
3. If the **hft** is emulated, then do the following:
   a. Set the pseudo-tty for raw data. That is, disable all input and output processing. This is necessary because the control sequences can contain binary data that would be misinterpreted by the pseudo-tty device driver as ASCII control codes. See "termio" on page 6-114 for details.

   b. Use the **write** system call to send an **hfctlreq** VTD structure, immediately followed by the request structure, if any, that would normally be pointed to by the **ioctl** *arg* parameter. The **hfctlreq** structure contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFCTLREQH**
hf_intro.hf_typelo	**HFCTLREQL**
hf_request	The request type.
hf_arg_len	The length of the argument structure that follows the **hfctlreq** VTD, or 0 if none.

hf_rsp_len The maximum length of the response data structure that is to be returned with the **hfctlack** VTD. This value is 0 if no response buffer is expected.

c. Read (using the **read** system call) until an acknowledgement VTD is received. This acknowledgement takes the form of an **hfctlack** structure, which is sometimes followed by a returned data structure, depending on the operation requested. The **hfctlack** structure contains the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFCTLACKH**
hf_intro.hf_typelo	**HFCTLACKL**
hf_request	The type of request that is being acknowledged.
hf_ret_code	The error code: zero indicates successful completion; a non-zero value is the value that is normally found in **errno**.
hf_arg_len	The length of the response data structure that follows the **hfctlack** VTD, or 0 if none. The length must not exceed the value of **hf_rsp_len** that was specified in the **hfctlreq** structure.

The file **/usr/lib/samples/hft/hftctl.c** contains a sample program that illustrates how to implement this procedure.

Input

Data read from an **hft** device with the **read** system call can contain not only character data entered from a keyboard, but also input from other devices, such as a locator, a tablet, valuators, and lighted programmable function keys. Data from devices other than the keyboard is passed back from the **read** system call in the form of special control sequences that are described in this section.

Note: These control sequences contain binary data. To prevent the binary data from being misinterpreted as ASCII control codes, Set the terminal's canonical processing off. See ICANON on page 6-121 for details.

Untranslated Key Control

If keyboard input is received when HFXLATKBD is turned off, this control sequence is returned. The key position identifies the logical key pressed. The key status bits indicate **Alt**, **Ctrl**, **Shift**, **Caps Lock**, and **Num Lock** key states. The scan code and make/break keys are dependent upon hardware and require knowledge of the physical keyboard in use. See “keyboard” on page 6-78 for additional information.

The structure of the untranslated key control is:

```
struct hfunxlate
{
    char hf_esc;
    char hf_lbr;
    char hf_ww;
    char hf_keypos;
    char hf_scancode;
    char hf_status[2];
};
```

The fields of the structure are:

Field	**Description**
hf_esc	ESC (0x1B)
hf_lbr	[(0x5B)
hf_ww	w (0x77)
hf_keypos	Key Position
hf_scancode	Scan Code (see “keyboard” on page 6-78)

hf_status[0] Status:

HFUXSHIFT	A shift key is pressed.
HFUXCTRL	**Ctrl** key is pressed.
HFUXALT	**Alt** key is pressed.
HFUXCAPS	**Caps Lock** mode is in effect.
HFUXNUM	**Num Lock** mode is in effect.
HFUXMAKE	If set, key has been pressed. If not set, key has been released.

hf_status[1] Status:

HFUXRPT	Automatic repeat (typematic) state
HFUXLSH	Left shift state
HFUXRSH	Right shift state
HFUXLALT	Left alternate shift state
HFUXRALT	Right alternate shift state.

Input Device Report

This control reports input data from the mouse, tablet, LPFKs, or valuator dials. The data is reported in the form of an **hflocator** structure, and the following sections describe the fields of this structure for each type of input device.

Mouse Report

hf_esc ESC (0x1B)

hf_lbr [(0x5B)

hf_why y (0x79)

hf_deltax The X delta, a signed integer that holds the relative X delta accumulations in counts of 0.25 millimeters of the locator movement in twos-complement form. This information is sent to the virtual terminal to indicate horizontal movement since the last locator movement.

hf_deltay The Y delta, a signed integer that holds the relative Y delta accumulations in counts of 0.25 millimeters of the locator movement in twos-complement form. This information is sent to the virtual terminal to indicate vertical movement since the last locator movement.

hf_seconds Time of the locator report in whole seconds since system startup.

hf_sixtyths The fractional part of time stamp of the locator report in 1/60th of seconds.

hf_buttons The status of the locator buttons. This information is sent to the virtual terminal to indicate a change in the status of the buttons since the last locator movement in the following manner:

HFBUTTON1	Button 1 has been pressed.
HFBUTTON2	Button 2 has been pressed.
HFBUTTON3	Button 3 has been pressed.

hf_stype 0

Tablet Report

hf_esc **ESC** (0x1B)

hf_lbr [(0x5B)

hf_why y (0x79)

hf_deltax The absolute X coordinate of the tablet sensor.

hf_deltay The absolute Y coordinate of the tablet sensor.

hf_seconds Time of the locator report in whole seconds since system startup.

hf_sixtyths The fractional part of time stamp of the locator report in 1/60th of seconds.

hf_buttons The status of the locator buttons. This information is sent to the virtual terminal to indicate a change in the status of the buttons since the last locator movement in the following manner:

HFBUTTON1	The left button has been pressed.
HFBUTTON2	The right button has been pressed.

hf_stype 1

LPFK Report

hf_esc **ESC** (0x1B)

hf_lbr [(0x5B)

hf_why y (0x79)

hf_deltax The LPFK number.

hf_deltay Reserved.

hf_seconds Time of the report in whole seconds since system startup.

hf_sixtyths The fractional part of time stamp of the report in 1/60th of seconds.

hf_buttons The status of the LPFK.

hf_stype 2

Valuator Dial Report

hf_esc ESC (0x1B)

hf_lbr [(0x5B)

hf_why y (0x79)

hf_deltax The dial number.

hf_deltay The dial value delta. This is a signed integer value in the dial's units of granularity (see "Set Dial Granularities" on page 6-66).

hf_seconds Time of the report in whole seconds since system startup.

hf_sixtyths The fractional part of time stamp of the report in 1/60th of seconds.

hf_buttons The status of the dial.

hf_stype 3

Adapter-Generated Input

Some adapters can return status information to MOM applications by way of a ring buffer. This status information is placed in the ring buffer with a VTA multi-byte control (ESC [r). This feature is not available to KSR mode.

The information that immediately follows the control sequence includes a 1-byte queue ID and 20 bytes of data. Note that the hardware returns 16-bit words and that the bit-numbering conventions are reversed. See *IBM RT PC Hardware Technical Reference* for details on the data returned for each adapter status entry.

Status	QID	Data
FIFO mode entered	0x01	0x03 in first data byte, rest reserved.
PHIGS traversal started	0x01	0x05 in first data byte, rest reserved.
FIFO pick mode set	0x01	0x07 in first data byte, rest reserved.
CGA mode entered	0x01	0x09 in first data byte, rest reserved.
Traversal stopped	0x01	0x0B in first data byte, rest reserved.
Single-step mode completed	0x01	0x0F in first data byte, rest reserved.

Status	QID	Data
Echo cursor completed	0x01	0x11 in first data byte, rest reserved.
Defined pointer echo completed	0x01	0x13 in first data byte, rest reserved.
Remove cursor completed	0x01	0x15 in first data byte, rest reserved.
Clear frame buffer completed	0x01	0x17 in first data byte, rest reserved.
Load look-up table completed	0x01	0x21 in first data byte, rest reserved.
Set pick window size completed	0x01	0x27 in first data byte, rest reserved.
Reset FIFO pick mode completed	0x01	0x29 in first data byte, rest reserved.
Set blink mode completed	0x01	0x2D in first data byte, rest reserved.
Reset blink mode completed	0x01	0x2F in first data byte, rest reserved.
Initialization complete	0x02	0x01 in first data byte, rest reserved.
Traversal complete	0x03	No data, all 20 bytes reserved.
Pick occurred	0x04	Data words 1-5 set to reason extension words 1-10.
Buffer error FIFO overflow Illegal graphic order Illegal request code Invalid page Stack error Traversal error	0x05	Data words 1-5 set to reason extension words 1-10.
PELPRO task completed	0x06	No data, all 20 bytes reserved.
PELPRO pick	0x07	Data words 1-5 set to reason extension words 1-10.
PELPRO vertical synch.	0x08	No data, all 20 bytes reserved.
FIFO half full	0x09	No data, all 20 bytes reserved.
FIFO half empty	0x0A	No data, all 20 bytes reserved.
Synchronize	0x0B	Data words 1-5 set to reason extension codes 1-10.

Output

ASCII data can be sent to the virtual terminal using the **write** system call along with data of any length. In addition, virtual terminal control structures are sent to the virtual terminal using the **write** system call.

Each control structure is introduced by a ***virtual terminal data*** (VTD) character sequence. The VTD prefix consists of the ASCII codes **ESC**, [, and **x** (0x1B5B78). This is followed by a length and an operation type code. The data that follows this structure depends on the type of control.

The **hfintro** structure looks like this:

```
{
    char hf_esc;
    char hf_lbr;
    char hf_ex;
    char hf_len[4];
    char hf_typehi;
    char hf_typelo;
};
```

The significant fields in the **hfintro** structure are:

hf_len The total number of bytes in the header and associated data, not including the three-character VTD control sequence. In other words, the length is the total number of characters in the control sequence minus 3.

hf_typehi The high-order byte of the information type code.

hf_typelo The low-order byte of the information type code.

Note that **hf_typehi** and **hf_typelo** are called the **major** and **minor** data types in *Virtual Resource Manager Technical Reference.* The values of **hf_typehi** and **hf_typelo** are documented with each command.

Because the **hfintro** structure is an odd number of bytes in length, it is designated as the character array **hf_intro[HFINTROSZ]** in the structures that define the various operation requests. This prevents the C compiler from inserting bytes into the structure to align the following fields on word boundaries. The **hf_typehi** and **hf_typelo** fields are referred to **hf_intro.hf_typehi** and **hf_intro.hf_typelo** in this book, although these references are not precisely correct.

All reserved and unused fields must be set to 0. You can set the entire structure to 0 and then fill in the appropriate fields.

Protocol Modes

Protocol modes determine how the virtual terminal will interpret coded data, translate and return input data. Two bits control each mode. The first, in the **hf_select** field, indicates whether to use the current mode setting. If this bit is set, then the corresponding bit in **hf_value** indicates the new setting for the mode. The mode bits are set to the default value when the virtual terminal is opened. These defaults may be changed during configuration with the **HFRCONF** operation.

The **hfprotocol** structure gives the protocol definitions:

Field	**Value**
hf_intro.hf_typehi	**HFKSRPROH** or **HFMOMPROH**
hf_intro.hf_typelo	**HFKSRPROL** or **HFMOMPROL**
hf_sublen	2
hf_subtype	0
hf_select	Specifies which modes to change. A bit value of 1 specifies the mode represented by that bit to change.
hf_select[0]	Mode selectors: **HFHOSTS** **HFXLATKBD**
hf_select[1]	Mode selectors: **HFWRAP** **HFLOCATOR** **HFLPFKS** **HFDIALS** **HFDINTR** **HFDINTRONLY**
hf_value[0]	New mode values: **HFHOSTS** **HFXLATKBD**
hf_value[1]	New mode values: **HFWRAP** **HFLOCATOR** **HFLPFKS** **HFDIALS** **HFDINTR** **HFDINTRONLY**

When issuing this command, specify a type of either **HFKSRPROH**, **HFKSRPROL** or **HFMOMPROH**, **HFMOMPROL** depending on whether you are sending this command from within Keyboard Send-Receive mode (KSR) or Monitor mode (MOM). Only certain protocol modes are valid in each of these modes, as shown in the following table. An attempt to set an invalid protocol mode is ignored.

Protocol Mode	**When Valid**	**Meaning**
HFHOSTS	KSR	A 0 bit (default) means not to report shift key depressions. A 1 bit means report shift key depressions. **HFHOSTS** mode specifies whether to report keyboard status changes. If **HFHOSTS** mode is set, the keyboard status information is returned in the KSI ANSI control (see "Multi-Byte Controls" on page 5-13).
HFXLATKBD	KSR, MOM	A 1 bit (default) specifies that the keyboard input is translated. A 0 bit indicates send key data as untranslated key controls. See "Untranslated Key Control" on page 6-56.
HFWRAP	KSR	A 1 bit (default) causes the cursor to wrap when the presentation space boundary is exceeded. A 0 bit specifies do not wrap the cursor.
HFLOCATOR	KSR, MOM	A 0 bit (default) disables the locator from sending data. A 1 bit enables the locator to send data.
HFLPFKS	KSR, MOM	A 0 bit (default) disables LPFK input. A 1 bit enables LPFK input.
HFDIALS	KSR, MOM	A 0 bit (default) disables dial (valuator) input. A 1 bit enables dial input.
HFDINTR	MOM	A 0 bit (default) indicates that display adapter status information is not to be sent to the host. A 1 bit specifies that the display status is to be sent.
HFDINTRONLY	MOM	A 0 bit (default) specifies not to restrict the use of the MOM input ring buffer. A 1 bit specifies to restrict the use of the MOM input ring buffer to display adapter status information only.

Set Keyboard LEDs

The structure for this command is **hfkled**, and it contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFKLEDCH**
hf_intro.hf_typelo	**HFKLEDCL**
hf_sublen	2
hf_subtype	1
hf_ledselect	Indicates which of three LEDs to change: **HFNUMLOCK** The **Num Lock** LED **HFCAPSLOCK** The **Caps Lock** LED **HFSCROLLOCK** The **Scroll Lock** LED.
hf_ledvalue	Indicates the value to which to set the LEDs specified in **hf_ledselect**. LEDs that are specified with a 1 bit are set: **HFNUMLOCK** The **Num Lock** LED **HFCAPSLOCK** The **Caps Lock** LED **HFSCROLLOCK** The **Scroll Lock** LED.

Set Locator Thresholds

The locator device receives notice of horizontal and vertical movement. The delta of these movement events are monitored by the driver, until the accumulated events exceed either the horizontal or vertical thresholds, or both. The locator device accumulates measurements at consecutive samplings. When a threshold is exceeded, the driver queues the information to the virtual terminal. When the status of the locator buttons change, the accumulated measurements are returned to the virtual terminal, even if these measurements do not exceed a threshold. The virtual terminal provides neither echoing nor positional management functions for the locator.

Each opened virtual terminal has its own threshold values. When a virtual terminal is opened, the threshold values default to 2.75 millimeters horizontal and 5.5 millimeters vertical. If the thresholds are 0, each event report is returned to the virtual terminal at the sampling rate supported by the locator device driver.

Setting the **HFLOCATOR** bit to 0 in the protocol mode definition or setting both thresholds to the maximum values completely disables the locator input.

The **hfloth** structure is used for the locator threshold command, and it contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFLOTHCH**
hf_intro.hf_typelo	**HFLOTHCL**
hf_sublen	2
hf_subtype	1
hf_hthresh	Specifies the horizontal threshold in values from 0 to 32767 in units of 0.25 millimeters.
hf_vthresh	Specifies the vertical threshold in values from 0 to 32767 in units of 0.25 millimeters.

Set Tablet Dead Zones

Dead zones are areas of the tablet from which no input reports are generated. Each virtual terminal can set its own dead zones.

Initially, both of the dead zone values are set to 0, making the entire tablet active. Setting both values to 32767 completely disables tablet input, as does turning off **HFLOCATOR** in the protocol mode definition (see "Protocol Modes" on page 6-62).

The **hftdzone** structure is used for this command, and it contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFTDZCH**
hf_intro.hf_typelo	**HFTDZCL**
hf_sublen	2
hf_subtype	1
hf_horizontal	A 2-byte nonnegative value specified in units of 0.25 millimeters.
hf_vertical	A 2-byte nonnegative value specified in units of 0.25 millimeters.

Set LPFKs

The **hfdial_lpfk** structure is used for this command, and it contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFLPFKSCH**
hf_intro.hf_typelo	**HFLPFKSCL**
hf_sublen	2

hf_subtype	1
hf_mask.keys	A 4-byte bit mask numbered 0 to 31. Bits that are set specify LPFK flag values to change.
hf_data2.lpfk.flags	A 4-byte set of bits numbered 0 to 31. For LPFKs selected by **hf_mask.keys**, a 0 bit disables the LPFK, and a 1 bit enables the LPFK.

Set Dial Granularities

The **hfdial_lpfk** structure is used for this command, and it contains the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFDIALSCH**
hf_intro.hf_typelo	**HFDIALSCL**
hf_sublen	2
hf_subtype	1
hf_mask.dials	A 4-byte bit mask numbered 0 to 31. Bits that are set specify dial granularity values to change.
hf_data2.granularity	An array of sixteen 1-byte values giving the granularity of each dial. Granularity is the number of events per full 360° revolution of the dial. The values in the array represent powers of 2, and they can range from 2 to 8.

Sound

This command sends output to the speaker. The mode byte determines whether to execute sound commands for the active virtual terminal and whether to interrupt the application after the sound command executes. No range check is made for the frequency or duration values. The **hfsound** structure is used for this command:

Field	**Value**
hf_intro.hf_typehi	**HFSOUNDCH**
hf_intro.hf_typelo	**HFSOUNDCL**
hf_sublen	2
hf_subtype	1
hf_mode	Mode: **HFSIGSOUND** If set, causes the **SIGSOUND** signal to be sent to the process when this sound command is executed or discarded. If not set, then no signal is sent.

HFEXECALWAYS
If set, causes this sound command to be executed whether or not this virtual terminal is active. If not set, then the sound command is executed only if the terminal is active.

hf_dur Duration in 1/128 seconds.

hf_freq Frequency in hertz.

Cancel Sound

The cancel sound command removes all commands from the speaker device that do not want sound commands executed. Only the commands that have the **execute all sound to this terminal** flag are left in the active terminal queue. An inactive terminal ignores this command.

Sending a **cancel** and/or **enable sound** command flushes the speaker driver queue when a virtual terminal transition occurs. Regardless of whether the sound request is executed or purged, the virtual terminal receives a response if the response flag is set (bit 0 of sound command byte 0 is equal to 1).

The **hfcansnd** structure is used for this command, and it contains the following fields:

Field	Value
hf_intro.hf_typehi	**HFCANSNDCH**
hf_intro.hf_typelo	**HFCANSNDCL**

Change Physical Display

This command changes the default physical display characteristics specified in the virtual terminal defaults.

The **hfchgdsp** structure is used for this command:

Field	Value
hf_intro.hf_typehi	**HFCHGDSPCH**
hf_intro.hf_typelo	**HFCHGDSPCL**
hf_sublen	2
hf_subtype	0
hf_mode	Bits 0-1 and 3-15 of these bytes are reserved. Bit 2 specifies the default or another value specified for the physical display.

HFNONDEF If set, uses the identifier specified in bytes 10-13 for the physical display. If not set, uses the physical screen defaults.

hf_rsvd1 Reserved

hf_devid Physical display device identifier.

hf_rsvd2 Reserved

Note: If the physical terminal is changed, it may be necessary to change the **TERM** environment variable. See "TERM" on page 5-72 and "terminfo" on page 4-148.

Keyboard Send-Receive Mode (KSR)

In KSR mode, each byte written to the virtual terminal is interpreted as an ASCII code, which can be a displayable character, a single-byte control, or part of an escape or control sequence. "data stream" on page 5-5 explains the supported ASCII/ANSI data stream in detail. KSR mode also supports a number of special control sequences specific to the virtual terminal environment.

A KSR virtual terminal has a presentation space (PS) of a fixed number of columns per line, and a fixed number of lines. A symbol can be placed at any column on any line in the presentation space. A pointer into the virtual terminal defines the cursor position with a column and a line number. Graphics from the KSR data stream are placed in the PS relative to the cursor position. Keyboard input also relates to the cursor position.

Two common modes for displaying graphics are ***replace*** and ***insert***. In replace mode, a graphic character sent to a KSR terminal is placed above the cursor, replacing the symbol already there. In insert mode, a graphic character sent to a KSR terminal is also placed above the cursor, but the symbol above the cursor and all symbols on the same line are shifted right one column position on the line. Characters shifted from the last column on the line disappear.

Another mode determines cursor movement after the the last column position of a line. This mode, ***automatic new line*** (AUTONL), determines if the cursor wraps around to the first column position of the next line or stays at the last column on the current line.

If AUTONL is set, the cursor moves to the first column position of the following line. If the cursor happens to be on the bottom line of the presentation space, the presentation space scrolls up one line. If AUTONL is reset, the cursor stays on the last column of the current line.

Blank lines in the presentation space and erased character positions display in the active background color with normal attributes.

To set the KSR protocol modes, write a protocol mode control, which is described under "Protocol Modes" on page 6-62. Specify the type as **HFKSRPROH, HFKSRPROL**.

The following control sequences are valid only in a KSR-mode data stream.

Character Set Definition

The ASCII character set-to-display code (font) mapping of a virtual terminal can be altered. For each virtual terminal, the virtual terminal maintains character set mapping tables for two unique user-definable character sets called ***Unique One*** and ***Unique Two***. These sets contain 256 ten-bit display symbol codes, and are activated by the SG0 or SG1 control (see "Multi-Byte Controls" on page 5-13).

Note: Data is kept in display symbol form in the virtual terminal, and translation back to ASCII codes is done using the standard character set definitions, not Unique One or Two.

The **hfcharset** structure is used for character set definition, and it contains the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFCHARSETH**
hf_intro.hf_typelo	**HFCHARSETL**
hf_sublen	2
hf_subtype	1
hf_setnum	User-defined character set **HFUNIQ1** Unique One (user-definable set 1) **HFUNIQ2** Unique Two (user-definable set 2)
hf_rsvd	Reserved
hf_code	10-bit display symbol code. This field may be repeated up to 256 times. See "display symbols" on page 5-24 for the values of the display symbols.

Set KSR Color Palette

This command specifies the color to associate with certain display adapters. The default color palettes are the ANSI 3.64 palette for character terminals and the PC color palette for all-points-addressable terminals. If the color specified is not supported by the adapter, the virtual display driver sets that color to the default for that mode.

The structure for this command is **hfcolorpal**, and it contains the following fields:

Field	**Value**
hf_intro.hf_typehi	**HFCOLORPALH**
hf_intro.hf_typelo	**HFCOLORPALL**
hf_sublen	2
hf_subtype	1
hf_numcolor	Number of entries in the palette
hf_palet	Adapter-specific settings of the first entry in the color palette. These settings must be repeated for each entry of the color palette corresponding to the display adapter. See *IBM RT PC Hardware Technical Reference* for information about the display adapter.

Change Fonts

When a virtual terminal is first opened, and whenever it is changed, this assignment is made to the first font in the list of configured fonts. The virtual terminal initially tries to select a font that results in a presentation space of 80 columns by 25 rows. The first font with a normal appearance (not italics) that meets this criteria is chosen. If no fonts meet this criteria, the first font that can be displayed on the particular device is selected. All alternate fonts will be initialized to this chosen ID.

Note that if the font is changed, the data currently in the presentation space is lost, and the cursor reverts to the double underscore and is placed at the home position (first column, first row). If it is desirable to control fonts, the fonts should be explicitly set when opening a terminal or changing a display.

If the **change fonts** request is accepted and the installed fonts are a different size than the previous fonts, the presentation space size is adjusted to the number of rows and columns that fit on the physical display screen for the new font size.

See the **/usr/lib/samples/README.font** file for information about defining and selecting fonts.

The **hffont** structure is used for this request:

Field	**Value**
hf_intro.hf_typehi	**HFFONTH**
hf_intro.hf_typelo	**HFFONTL**
hf_sublen	2
hf_subtype	1
hf_primary	Physical font ID of primary font attribute.
hf_alt1	Physical font ID of first alternate font attribute.
hf_alt2	Physical font ID of second alternate font attribute.
hf_alt3	Physical font ID of third alternate font attribute.
hf_alt4	Physical font ID of fourth alternate font attribute.
hf_alt5	Physical font ID of fifth alternate font attribute.
hf_alt6	Physical font ID of sixth alternate font attribute.
hf_alt7	Physical font ID of seventh alternate font attribute.

Cursor Representation

The cursor representation data format determines how the cursor is presented on the display screen. The **hfcursor** structure is used for this request:

Field	**Value**	
hf_intro.hf_typehi	**HFCURSORH**	
hf_intro.hf_typelo	**HFCURSORL**	
hf_sublen	2	
hf_subtype	0	
hf_rsvd	Reserved.	
hf_shape	Cursor shape:	
	HFNONE	No cursor.
	HFSINGLUS	Single underscore.
	HFDBLUS	Double underscore.
	HFHALFBLOB	Lower half of illuminated character cell.
	HFMIDLINE	Double mid-character line.
	HFFULLBLOB	Full illuminated character cell.

Monitor Mode (MOM)

Programs that choose to interact more efficiently with a virtual terminal or that must operate the display in all-points-addressable mode should select the monitor mode of the virtual terminal. In this mode, the program performs output directly to the display adapter via a memory mapped I/O bus, thus avoiding write system calls. Such a program can optionally read data from a circular buffer, thus avoiding **read** system calls. Some execution speed is gained by operating in this mode, but portability is sacrificed because the program depends on specific display adapters.

Notes:

1. Do not leave terminal open in monitor mode.
2. No more than 1 process should be open to a virtual terminal in monitor mode.

In order for a user program to switch from normal KSR mode to monitor mode, it must perform several mode changes, which are accomplished using system calls. The display-sharing concept using virtual terminals causes the program in monitor mode to participate in the **next window** function by temporarily releasing the display. This is also accomplished using system calls. While the user program is active to the display, it performs output operations directly to the display hardware with memory mapped I/O ports.

Entering Monitor Mode

The first mode change the user program should perform is to request I/O bus access mode by opening the bus access pseudo device. See "bus" on page 6-5 for more information.

The next mode change that must be performed is to issue the **HFSMON ioctl** operation to enable monitor mode signals **SIGGRANT** and **SIGRETRACT**, and to specify the method that processes are to receive the signals. (See "Enter Monitor Mode (HFSMON)" on page 6-49.)

Next, the program should write a protocol mode control, which is described under "Protocol Modes" on page 6-62, specifing the type **HFMOMPROH, HFMOMPROL.**

The virtual terminal is now in monitor mode.

Only certain controls are valid for the **write** system call while in monitor mode. All other ASCII codes and controls are ignored. The valid controls and VTDs are:

- Disable Manual Input (DMI)
- Enable Manual Input (EMI)
- Set Keyboard LEDs
- Set Locator Threshold
- Set Locator Dead Zones
- Set Dials
- Sound
- Cancel Sound
- KSR Protocol

- MOM Protocol
- Screen Request
- Screen Release.

Screen Request and Input Ring Buffer Definition

Although the virtual terminal is in monitor mode, the program can perform direct operations on the display hardware only when granted permission by the operating system. The program first writes a screen request control.

This request uses the **hfmomscreq** structure, which contains the following fields:

Field	Value
hf_intro.hf_len	The length of the request up to and including the ring buffer.
hf_intro.hf_typehi	**HFMOMREQH**
hf_intro.hf_typelo	**HFMOMREQL**
hf_sublen	2
hf_subtype	0
hf_ringlen[2]	Shows the length of the ring in bytes.
hf_ringoffset[4]	Shows the offset to the input buffer ring (offset from the **hf_ringlen** field).

The **hf_ringlen** field specifies the size of the structure including the pointers and status fields. The program can directly access input key, locator, LPFK, and valuator data contained in the buffer without issuing **read** system calls.

The ring buffer structure (**hfmomring**, defined following) can be at any location in memory aligned on a word boundary. **hf_ringoffset** is the difference between the ring buffer address and the address of **hf_ringlen**, and it must be a positive value. Usually, the **hfmomring** ring buffer structure is defined so that it immediately follows the **hfmomscreq** structure in memory. Note that the compiler may implicitly insert one or more filler bytes between the two structures to align them at a memory address boundary. The value of **hf_ringoffset** must reflect such filler bytes. See the **/usr/lib/samples/hft/hftmom.c** source file for an example of how to calcuate **hf_ringoffset**.

If you do not want to specify or use a ring buffer, then set the **hf_len** field of the **hf_intro** to the size of only the introducer. In this case, read input with the standard **read** system call.

```
struct hfmomring
{
   char hf_rsvd[2];
   char hf_intreq;
```

```
        char hf_ovflow;
        unsigned hf_source;
        unsigned hf_sink;
        int  hf_unused[5];
        char hf_rdata[HFRDATA];
    };
```

The fields in this structure are defined as:

Field **Value**

hf_rsvd Reserved.

hf_intreq Interrupt request can be set to 0xFF by the application to cause the virtual terminal subsystem to send a **SIGMSG** signal each time an input event occurs. If this flag is set to 0 (the default), then a signal is sent to the application only when the buffer goes from being empty to nonempty. This byte is automatically reset to 0 by the virtual terminal each time it stores input data into the ring buffer. See "Reading Input Data from the Ring Buffer" for further discussion.

hf_ovflow Overflow determines whether the input buffer ring can accommodate more input information. A value of 0xFF indicates an overflow; 0x00 indicates normal operation.

hf_source Ring offset for virtual terminal represents the offset into the input ring where the virtual terminal queues keyboard and locator input. This offset starts from the beginning of the ring, so the absolute minimum value for the virtual terminal offset is 32 bytes. Application programs must not alter this field. If a program attempts to alter it, then the virtual terminal is killed.

hf_sink Ring offset for application shows the offset into the input ring from which the application reads keyboard and locator information from the event queue. This offset also starts from the beginning of the input ring, so the minimum value for this offset is 32 bytes.

hf_unused Reserved.

Reading Input Data from the Ring Buffer

The program should start the offsets **hf_source** and **hf_sink** to be equal. This indicates buffer empty condition. The program should then issue the **pause** system call, waiting for input. When the buffer goes from being empty to not empty, the program receives a **SIGMSG** signal. (Note that sending the **hfmomscreq** structure and defining the input ring buffer enables the sending of this signal.) The program should extract characters from the ring buffer while incrementing the **hf_sink** offset for each character extracted, making sure to wrap around after reaching the end of the buffer. Care should be taken to ensure the buffer empty condition is properly detected. The program should test the equality of the offsets after it has updated the **hf_sink** offset. Therefore, the order of operation is: extract a character, update the offset in its memory location, and test the equality of offsets; if the offsets are equal, then set **hf_intreq** to 0xFF.

IF **hf_source = = hf_sink - 1** (modulo ring size), then the ring buffer is full. If **hf_ovflow = = 0xff**, then an overflow condition exists. The overflow condition indicates input data has been lost. The program resets the overflow condition by clearing **hf_ovflow**.

Certain keys can be designated so they can be obtained using the **read** system call. This is particularly useful when such keys are the **Int** and **Quit** keys (see "termio" on page 6-114). These keys are designated using HFSECHO. Thus, by designating these keys in the break map, and by setting the ISIG mode of **termio**, it is possible to asynchronously interrupt a monitor mode program by pressing one of these keys.

Next Window Function

If a virtual terminal in monitor mode is active, pressing the Next Window key causes a SIGRETRACT signal to be sent to the process or group of processes specified by the HFSMON type **ioctl** system call. Before activating the next virtual terminal, the operating system allows the program a chance to save the state of the display hardware, such as registers and refresh memory. After this is done, the program should write a screen release control to the terminal to inform the operating system the state of the display hardware can be changed.

The screen release control is given by the **hfmomscrel** structure:

Field	**Value**
hf_intro.hf_len	The length of the entire structure, including the ring buffer, minus 3
hf_intro.hf_typehi	**HFMOMRELH**
hf_intro.hf_typelo	**HFMOMRELL**

After the display is released, the next virtual terminal is activated. If this is not done within 30 seconds of the receipt of the **SIGRETRACT** signal, all processes in that terminal group receive a **SIGKILL** signal. This is a safeguard to prevent disabled programs from disrupting the next window function.

The program can issue a **pause** system call if there is no work to do while the display is not available. When the monitor mode virtual terminal is activated again with the Next Window key, the program receives a **SIGGRANT** signal. In other words, the the program can resume direct output to the display. The display hardware state cannot be assumed to be the same as when the program released it.

Exiting Monitor mode

When the program has no further use of the monitor mode, it should first write a screen release control, followed by a KSR protocol control. This is especially important if the virtual terminal is open by another process, such as the parent process, which is often the command shell. If the program is certain that no other processes have the terminal open, it can simply issue a **close** system call to remove that virtual terminal.

Next, an **HFCMON ioctl** operation should be issued to make sure that no monitor mode signals have been sent to this process or other process in the terminal group in error.

Signals

In addition to the standard terminal signals (**SIGINT** and **SIGQUIT**), the virtual terminal generates other unique signals to inform the application program of asynchronous events. These signals include:

SIGGRANT Informs the user program that the display hardware can be directly accessed. This signal is sent following a monitor mode screen request VTD sequence. It is also sent after a monitor mode terminal has been made active with the next window key.

SIGRETRACT Informs the user program that the display hardware must be released for use by another program. This signal is sent after a monitor terminal being made inactive with the next window key.

SIGKILL Sent to all processes in the terminal **tty** group to enforce the **SIGRETRACT** signal. If the user program does not respond with a screen release VTD sequence within 30 second after receiving a **SIGRETRACT** signal, the **SIGKILL** terminates all processes associated with that virtual terminal and the terminal is closed.

SIGMSG Informs the user program that data has been placed into a previously empty input buffer.

Files

/dev/hft/*

Related Information

In this book: "ioctl" on page 2-56, "fonts" on page 4-68, "data stream" on page 5-5, "config" on page 6-7, and "termio" on page 6-114.

The discussion of the virtual terminal subsystem in *Virtual Resource Manager Technical Reference.*

Keyboard Description and Character Reference.

keyboard

Purpose

Maps the 101-Key RT PC keyboard.

Description

A keyboard mapping table is maintained for each virtual terminal. This table relates a key indicated by its key position along with the shift, control, or alternate keys to a character, mode processor function or string of characters. Portions or all of this mapping table can be modified by data passed to the **hfbuf** structure in the HFSKBD type **ioctl** system call. See "hft" on page 6-23 for information about this **ioctl** system call. See *Keyboard Description and Character Reference* for details about other RT PC keyboards.

Each key on the standard RT PC keyboard has a numeric position code that is used for this field. Figure 6-3 matches the key to its position code.

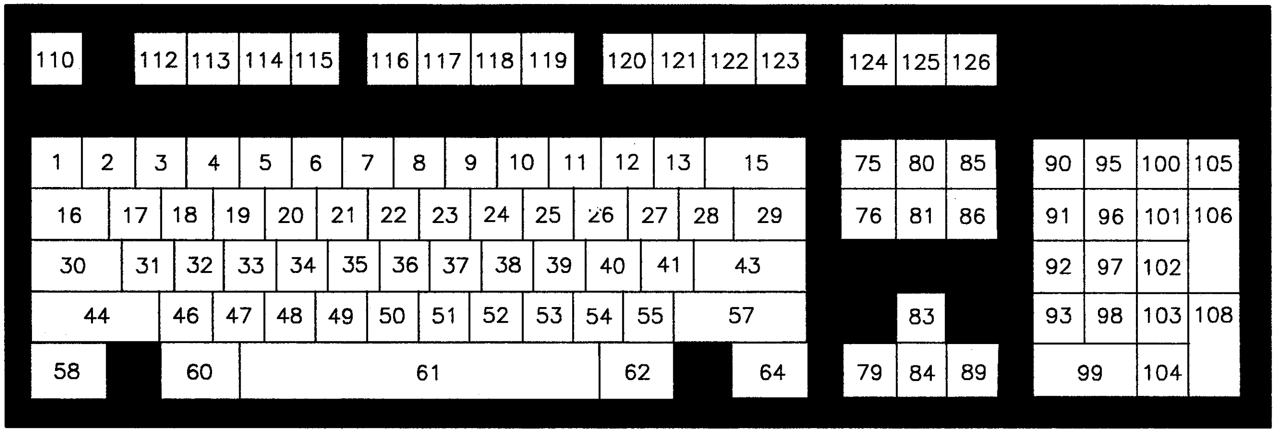


Figure 6-3. Position Codes for Remapping a Keyboard

The following keys are not redefinable because their function is predefined at the VRM level.

Key Position	**Function**	**States that cannot be remapped**
30	Caps Lock key	All states
44	Left Shift key	All states
57	Right Shift key	All states
58	Control key	All states
60	Left Alternate key	All states
62	Right Alternate key	All states
64	Action key	Shift, Control, Alternate, and Alternate Graphics states
90	Num Lock key	Base and Shift states

US 101-Key Keyboard Translate Table

The following table gives this information about the default mapping for the U.S. 101-key keyboard:

Key Posn – Keyboard key position.

Shift – The shift state of the position: Base, Shift, Ctrl, or Alt. (Note that the Alt Gr shift state is always the same as the Alt state.)

Assignment – The character or control assigned to that key.

Keyboard Definition – Provides information that as it would appear as part of a keyboard definition structure. (See "Set Keyboard Map (HFSKBD)" on page 6-36 for details.) Within the table, interpret the fields as follows:

nnn – One-byte key position number being defined.

s – Shift state being defined:

b (base) – No **Shift** key is pressed.

s (shift) – Either left or right **Shift** key is pressed.

c (control) – **Ctrl** key is pressed.

a (alt) – **Alt** key is pressed.

t – Type of definition:

c – Regular character definition, followed by a 1-byte code page identifier and a 1-byte code point specification. The code page identifiers are:

> for Code Page P0
= for Code Page P1
> for Code Page P2

This identifier is followed by a 1-byte code point identifier, given in the table as a decimal number.

f – Function specification, followed by a 2-byte function identifier, which is specified in the table as a hexidecimal value.

s – String specification, followed by a 1-byte code page identifier, a 1-byte string length and the 1-byte code point identifiers within the specified code page. No string specifications are included in the default keyboard layouts.

d – Nonspacing or ***dead*** character definition, followed by a 1-byte code page identifier and a 1-byte code point specification. The code page identifiers are as described for character definition. This is followed by a 1-byte code point identifier, given in the table as a decimal number.

Returned String – Specifies the data that is returned to the program that is reading the keyboard.

Notes – Specifies additional information. Entries in this column include:

- **CL** – This key is affected by **Caps Lock**.
- **DK** – This key is a dead character on this key state.
- **P1** – This key is a character from Code Page P1.
- **P2** – This key is a character from Code Page P2.

The **Alt** key, followed by one or more numbered keys on the numeric pad, will return a single character which has the value entered on the numeric pad. The value accumulates while the Alt key is held down and returns when that key is released. Only spacing character codes and single-byte controls are produced by this method.

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
1	Base	`	Grave Accent	1	b	c	< 96	0x60	
1	Shift	~	Tilde Accent	1	s	c	< 126	0x7e	
1	Ctrl	PFK	57	1	c	f	0x39	ESC [0 5 7 q	
1	Alt	PFK	115	1	a	f	0x73	ESC [1 1 5 q	
2	Base	1	One	2	b	c	< 49	0x31	
2	Shift	!	Exclamation Point	2	s	c	< 33	0x21	
2	Ctrl	PFK	49	2	c	f	0x31	ESC [0 4 9 q	
2	Alt	PFK	58	2	a	f	0x3a	ESC [0 5 8 q	
3	Base	2	Two	3	b	c	< 50	0x32	
3	Shift	@	At Sign	3	s	c	< 64	0x40	
3	Ctrl	NUL	Null	3	c	c	< 0	0x00	
3	Alt	PFK	59	3	a	f	0x3b	ESC [0 5 9 q	
4	Base	3	Three	4	b	c	< 51	0x33	
4	Shift	#	Number Sign	4	s	c	< 35	0x23	
4	Ctrl	PFK	50	4	c	f	0x32	ESC [0 5 0 q	
4	Alt	PFK	60	4	a	f	0x3c	ESC [0 6 0 q	
5	Base	4	Four	5	b	c	< 52	0x34	
5	Shift	$	Dollar Sign	5	s	c	< 36	0x24	
5	Ctrl	PFK	51	5	c	f	0x33	ESC [0 5 1 q	
5	Alt	PFK	61	5	a	f	0x3d	ESC [0 6 1 q	
6	Base	5	Five	6	b	c	< 53	0x35	
6	Shift	%	Percent Sign	6	s	c	< 37	0x25	
6	Ctrl	PFK	52	6	c	f	0x34	ESC [0 5 2 q	
6	Alt	PFK	62	6	a	f	0x3e	ESC [0 6 2 q	
7	Base	6	Six	7	b	c	< 54	0x36	
7	Shift	^	Circumflex Accent	7	s	c	< 94	0x5e	
7	Ctrl	SS2	Single Shift 2	7	c	c	< 30	0x1e	
7	Alt	PFK	63	7	a	f	0x3f	ESC [0 6 3 q	
8	Base	7	Seven	8	b	c	< 55	0x37	
8	Shift	&	Ampersand	8	s	c	< 38	0x26	
8	Ctrl	PFK	53	8	c	f	0x35	ESC [0 5 3 q	
8	Alt	PFK	64	8	a	f	0x40	ESC [0 6 4 q	

Key Posn	Shift State	Assignment		Keyboard Definition nnn	s	t		Returned String	Notes
9	Base	8	Eight	9	b	c	< 56	0x38	
9	Shift	*	Asterisk	9	s	c	< 42	0x2a	
9	Ctrl	PFK	54	9	c	f	0x36	ESC [0 5 4 q	
9	Alt	PFK	65	9	a	f	0x41	ESC [0 6 5 q	
10	Base	9	Nine	10	b	c	< 57	0x39	
10	Shift	(	Left Parenthesis	10	s	c	< 40	0x28	
10	Ctrl	PFK	55	10	c	f	0x37	ESC [0 5 5 q	
10	Alt	PFK	66	10	a	f	0x42	ESC [0 6 6 q	
11	Base	0	Zero	11	b	c	< 48	0x30	
11	Shift	)	Right Parenthesis	11	s	c	< 41	0x29	
11	Ctrl	PFK	56	11	c	f	0x38	ESC [0 5 6 q	
11	Alt	PFK	67	11	a	f	0x43	ESC [0 6 7 q	
12	Base	-	Hyphen	12	b	c	< 45	0x2d	
12	Shift	_	Underscore	12	s	c	< 95	0x5f	
12	Ctrl	SS1	Single Shift 1	12	c	c	< 31	0x1f	
12	Alt	PFK	68	12	a	f	0x44	ESC [0 6 8 q	
13	Base	=	Equal Sign	13	b	c	< 61	0x3d	
13	Shift	+	Plus Sign	13	s	c	< 43	0x2b	
13	Ctrl	PFK	69	13	c	f	0x45	ESC [0 6 9 q	
13	Alt	PFK	70	13	a	f	0x46	ESC [0 7 0 q	
14	Not available on keyboard								
15	Base	BS	Back Space	15	b	c	< 8	0x08	
15	Shift	BS	Back Space	15	s	c	< 8	0x08	
15	Ctrl	DEL	Delete	15	c	c	< 127	0x7f	
15	Alt	PFK	71	15	a	f	0x47	ESC [0 7 1 q	
16	Base	HT	Horizontal Tab	16	b	c	< 9	0x09	
16	Shift	CBT	Cursor Back Tab	16	s	f	0x105	ESC [Z	
16	Ctrl	PFK	72	16	c	f	0x48	ESC [0 7 2 q	
16	Alt	PFK	73	16	a	f	0x49	ESC [0 7 3 q	
17	Base	q	Lowercase q	17	b	c	< 113	0x71	CL
17	Shift	Q	Uppercase q	17	s	c	< 81	0x51	
17	Ctrl	DC1	Device Control 1	17	c	c	< 17	0x11	
17	Alt	PFK	74	17	a	f	0x4a	ESC [0 7 4 q	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
18	Base	w	Lowercase w	18	b	c	< 119	0x77	CL
18	Shift	W	Uppercase w	18	s	c	< 87	0x57	
18	Ctrl	ETB	End Trans Block	18	c	c	< 23	0x17	
18	Alt	PFK	75	18	a	f	0x4b	ESC [0 7 5 q	
19	Base	e	Lowercase e	19	b	c	< 101	0x65	CL
19	Shift	E	Uppercase e	19	s	c	< 69	0x45	
19	Ctrl	ENQ	Enquiry	19	c	c	< 5	0x05	
19	Alt	PFK	76	19	a	f	0x4c	ESC [0 7 6 q	
20	Base	r	Lowercase r	20	b	c	< 114	0x72	CL
20	Shift	R	Uppercase r	20	s	c	< 82	0x52	
20	Ctrl	DC2	Device Control 2	20	c	c	< 18	0x12	
20	Alt	PFK	77	20	a	f	0x4d	ESC [0 7 7 q	
21	Base	t	Lowercase t	21	b	c	< 116	0x74	CL
21	Shift	T	Uppercase t	21	s	c	< 84	0x54	
21	Ctrl	DC4	Device Control 4	21	c	c	< 20	0x14	
21	Alt	PFK	78	21	a	f	0x4e	ESC [0 7 8 q	
22	Base	y	Lowercase y	22	b	c	< 121	0x79	CL
22	Shift	Y	Uppercase y	22	s	c	< 89	0x59	
22	Ctrl	EM	End of Media	22	c	c	< 25	0x19	
22	Alt	PFK	79	22	a	f	0x4f	ESC [0 7 9 q	
23	Base	u	Lowercase u	23	b	c	< 117	0x75	CL
23	Shift	U	Uppercase u	23	s	c	< 85	0x55	
23	Ctrl	NAK	Not Acknowledge	23	c	c	< 21	0x15	
23	Alt	PFK	80	23	a	f	0x50	ESC [0 8 0 q	
24	Base	i	Lowercase i	24	b	c	< 105	0x69	CL
24	Shift	I	Uppercase i	24	s	c	< 73	0x49	
24	Ctrl	HT	Horizontal Tab	24	c	c	< 9	0x09	
24	Alt	PFK	81	24	a	f	0x51	ESC [0 8 1 q	
25	Base	o	Lowercase o	25	b	c	< 111	0x6f	CL
25	Shift	O	Uppercase o	25	s	c	< 79	0x4f	
25	Ctrl	SI	Shift In	25	c	c	< 15	0x0f	
25	Alt	PFK	82	25	a	f	0x52	ESC [0 8 2 q	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
26	Base	p	Lowercase p	26	b	c	< 112	0x70	CL
26	Shift	P	Uppercase p	26	s	c	< 80	0x50	
26	Ctrl	DLE	Data Link Enabl	26	c	c	< 16	0x10	
26	Alt	PFK	83	26	a	f	0x53	ESC [0 8 3 q	
27	Base	[	Left Bracket	27	b	c	< 91	0x5b	
27	Shift	{	Left Brace	27	s	c	< 123	0x7b	
27	Ctrl	ESC	Escape	27	c	c	< 27	0x1b	
27	Alt	PFK	84	27	a	f	0x54	ESC [0 8 4 q	
28	Base	]	Right Bracket	28	b	c	< 93	0x5d	
28	Shift	}	Right Brace	28	s	c	< 125	0x7d	
28	Ctrl	SS3	Single Shift 3	28	c	c	< 29	0x1d	
28	Alt	PFK	85	28	a	f	0x55	ESC [0 8 5 q	
29	Base	\	Reverse Slash	29	b	c	< 92	0x5c	
29	Shift	¦	Pipe Symbol	29	s	c	< 124	0x7c	
29	Ctrl	SS4	Single Shift 4	29	c	c	< 28	0x1c	
29	Alt	PFK	86	29	a	f	0x56	ESC [0 8 6 q	
30	Base	Caps Lock		None				Not Returned	
30	Shift	Caps Lock		None				Not Returned	
30	Ctrl	Caps Lock		None				Not Returned	
30	Alt	Caps Lock		None				Not Returned	
31	Base	a	Lowercase a	31	b	c	< 97	0x61	CL
31	Shift	A	Uppercase a	31	s	c	< 65	0x41	
31	Ctrl	SOH	Start of Header	31	c	c	< 1	0x01	
31	Alt	PFK	87	31	a	f	0x57	ESC [0 8 7 q	
32	Base	s	Lowercase s	32	b	c	< 115	0x73	CL
32	Shift	S	Uppercase s	32	s	c	< 83	0x53	
32	Ctrl	DC3	Device Control 3	32	c	c	< 19	0x13	
32	Alt	PFK	88	32	a	f	0x58	ESC [0 8 8 q	
33	Base	d	Lowercase d	33	b	c	< 100	0x64	CL
33	Shift	D	Uppercase d	33	s	c	< 68	0x44	
33	Ctrl	EOT	End of Transmission	33	c	c	< 4	0x04	
33	Alt	PFK	89	33	a	f	0x59	ESC [0 8 9 q	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
34	Base	f	Lowercase f	34	b	c	< 102	0x66	CL
34	Shift	F	Uppercase f	34	s	c	< 70	0x46	
34	Ctrl	ACK	Acknowledge	34	c	c	< 6	0x06	
34	Alt	PFK	90	34	a	f	0x5a	ESC [0 9 0 q	
35	Base	g	Lowercase g	35	b	c	< 103	0x67	CL
35	Shift	G	Uppercase g	35	s	c	< 71	0x47	
35	Ctrl	BEL	Bell	35	c	c	< 7	0x07	
35	Alt	PFK	91	35	a	f	0x5b	ESC [0 9 1 q	
36	Base	h	Lowercase h	36	b	c	< 104	0x68	CL
36	Shift	H	Uppercase h	36	s	c	< 72	0x48	
36	Ctrl	BS	Backspace	36	c	c	< 8	0x08	
36	Alt	PFK	92	36	a	f	0x5c	ESC [0 9 2 q	
37	Base	j	Lowercase j	37	b	c	< 106	0x6a	CL
37	Shift	J	Uppercase j	37	s	c	< 74	0x4a	
37	Ctrl	LF	Line Feed	37	c	c	< 10	0x0a	
37	Alt	PFK	93	37	a	f	0x5d	ESC [0 9 3 q	
38	Base	k	Lowercase k	38	b	c	< 107	0x6b	CL
38	Shift	K	Uppercase k	38	s	c	< 75	0x4b	
38	Ctrl	VT	Vertical Tab	38	c	c	< 11	0x0b	
38	Alt	PFK	94	38	a	f	0x5e	ESC [0 9 4 q	
39	Base	l	Lowercase l	39	b	c	< 108	0x6c	CL
39	Shift	L	Uppercase l	39	s	c	< 76	0x4c	
39	Ctrl	FF	Form Feed	39	c	c	< 12	0x0c	
39	Alt	PFK	95	39	a	f	0x5f	ESC [0 9 5 q	
40	Base	;	Semicolon	40	b	c	< 59	0x3b	
40	Shift	:	Colon	40	s	c	< 58	0x3a	
40	Ctrl	PFK	96	40	c	f	0x60	ESC [0 9 6 q	
40	Alt	PFK	97	40	a	f	0x61	ESC [0 9 7 q	
41	Base	′	Quote, Apostrophe	41	b	c	< 39	0x27	
41	Shift	″	Double Quote	41	s	c	< 34	0x22	
41	Ctrl	PFK	98	41	c	f	0x62	ESC [0 9 8 q	
41	Alt	PFK	99	41	a	f	0x63	ESC [0 9 9 q	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
42	Not available on keyboard								
43	Base	CR	Carriage Return	43	b	c	< 13	0x0d	
43	Shift	CR	Carriage Return	43	s	c	< 13	0x0d	
43	Ctrl	CR	Carriage Return	43	c	c	< 13	0x0d	
43	Alt	PFK	100	43	a	f	0x64	ESC [1 0 0 q	
44	Base	Shift (Left)		None				Not Returned	
44	Shift	Shift (Left)		None				Not Returned	
44	Ctrl	Shift (Left)		None				Not Returned	
44	Alt	Shift (Left)		None				Reserved for IBM 5080	
45	Not available on keyboard								
46	Base	z	Lowercase z	46	b	c	< 122	0x7a	CL
46	Shift	Z	Uppercase z	46	s	c	< 90	0x5a	
46	Ctrl	SUB	Substitute Char.	46	c	c	< 26	0x1a	
46	Alt	PFK	101	46	a	f	0x65	ESC [1 0 1 q	
47	Base	x	Lowercase x	47	b	c	< 120	0x78	CL
47	Shift	X	Uppercase x	47	s	c	< 88	0x58	
47	Ctrl	CAN	Cancel	47	c	c	< 24	0x18	
47	Alt	PFK	102	47	a	f	0x66	ESC [1 0 2 q	
48	Base	c	Lowercase c	48	b	c	< 99	0x63	CL
48	Shift	C	Uppercase c	48	s	c	< 67	0x43	
48	Ctrl	ETX	End of Text	48	c	c	< 3	0x03	
48	Alt	PFK	103	48	a	f	0x67	ESC [1 0 3 q	
49	Base	v	Lowercase v	49	b	c	< 118	0x76	CL
49	Shift	V	Uppercase v	49	s	c	< 86	0x56	
49	Ctrl	SYN	Synch Idle	49	c	c	< 22	0x16	
49	Alt	PFK	104	49	a	f	0x68	ESC [1 0 4 q	
50	Base	b	Lowercase b	50	b	c	< 98	0x62	CL
50	Shift	B	Uppercase b	50	s	c	< 66	0x42	
50	Ctrl	STX	Start of Text	50	c	c	< 2	0x02	
50	Alt	PFK	105	50	a	f	0x69	ESC [1 0 5 q	
51	Base	n	Lowercase n	51	b	c	< 110	0x6e	CL
51	Shift	N	Uppercase n	51	s	c	< 78	0x4e	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
51	Ctrl	SO	Shift Out	51	c	c	< 14	0x0e	
51	Alt	PFK	106	51	a	f	0x65	ESC [1 0 6 q	
52	Base	m	Lowercase m	52	b	c	< 109	0x6d	CL
52	Shift	M	Uppercase m	52	s	c	< 77	0x4d	
52	Ctrl	CR	Carriage Return	52	c	c	< 13	0x0d	
52	Alt	PFK	107	52	a	f	0x66	ESC [1 0 7 q	
53	Base	,	Comma	53	b	c	< 44	0x2c	
53	Shift	<	Less Than Sign	53	s	c	< 60	0x3c	
53	Ctrl	PFK	108	53	c	f	0x6c	ESC [1 0 8 q	
53	Alt	PFK	109	53	a	f	0x6d	ESC [1 0 9 q	
54	Base	.	Period	54	b	c	< 46	0x2e	
54	Shift	>	Greater Than Sign	54	s	c	< 62	0x3e	
54	Ctrl	PFK	110	54	c	f	0x6e	ESC [1 1 0 q	
54	Alt	PFK	111	54	a	f	0x6f	ESC [1 1 1 q	
55	Base	/	Slash	55	b	c	< 47	0x2f	
55	Shift	?	Question Mark	55	s	c	< 63	0x3f	
55	Ctrl	PFK	112	55	c	f	0x70	ESC [1 1 2 q	
55	Alt	PFK	113	55	a	f	0x71	ESC [1 1 3 q	
56	Not available on keyboard								
57	Base	Shift (Right)		None				Not Returned	
57	Shift	Shift (Right)		None				Not Returned	
57	Ctrl	Shift (Right)		None				Not Returned	
57	Alt	Shift (Right)		None				Reserved for IBM 5080	
58	Base	Control		None				Not Returned	
58	Shift	Control		None				Not Returned	
58	Ctrl	Control		None				Not Returned	
58	Alt	Control		None				Not Returned	
59	Not available on keyboard								
60	Base	Alternate Shift		None				Not Returned	
60	Shift	Alternate Shift		None				Not Returned	
60	Ctrl	Alternate Shift		None				Not Returned	
60	Alt	Alternate Shift		None				Not Returned	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
61	Base	SP	Space	61	b	c	< 32	0x20	
61	Shift	SP	Space	61	s	c	< 32	0x20	
61	Ctrl	SP	Space	61	c	c	< 32	0x20	
61	Alt	SP	Space	61	a	c	< 32	0x20	
62	Base	Alternate Graphic Shift		None				Not Returned	
62	Shift	Alternate Graphic Shift		None				Not Returned	
62	Ctrl	Alternate Graphic Shift		None				Not Returned	
62	Alt	Alternate Graphic Shift		None				Not Returned	
63	Not available on keyboard								
64	Base	PFK	114	64	b	f	0x72	ESC [1 1 4 q	
64	Shift	VTRM	Previous Window	None				VTRM Previous Window	
64	Ctrl	VTRM	Windows Window	None				VTRM Windows Window	
64	Alt	VTRM	Next Window	None				VTRM Next Window	
65 - 74	Not available on keyboard								
75	Base	PFK	139 INS Toggle	75	b	f	0x8b	ESC [1 3 9 q	
75	Shift	PFK	139 INS Toggle	75	s	f	0x8b	ESC [1 3 9 q	
75	Ctrl	PFK	140	75	c	f	0x8c	ESC [1 4 0 q	
75	Alt	PFK	141	75	a	f	0x8d	ESC [1 4 1 q	
76	Base	DCH	Delate Character	76	b	f	0x151	ESC [P	
76	Shift	DCH	Delate Character	76	s	f	0x151	ESC [P	
76	Ctrl	PFK	142	76	c	f	0x8e	ESC [1 4 2 q	
76	Alt	DL	Delete Line	76	a	f	0x153	ESC [M	
77	Not available on keyboard								
78	Not available on keyboard								
79	Base	CUB	Cursor Back	79	b	f	0x104	ESC [D	
79	Shift	PFK	158	79	s	f	0x9e	ESC [1 5 8 q	
79	Ctrl	PFK	159	79	c	f	0x9f	ESC [1 5 9 q	
79	Alt	PFK	160	79	a	f	0xa0	ESC [1 6 0 q	
80	Base	HOME		80	b	f	0x108	ESC [H	
80	Shift	PFK	143	80	s	f	0x8f	ESC [1 4 3 q	
80	Ctrl	PFK	144	80	c	f	0x90	ESC [1 4 4 q	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
80	Alt	PFK	145	80	a	f	0x91	ESC [1 4 5 q	
81	Base	PFK	146	81	b	f	0x92	ESC [1 4 6 q	
81	Shift	PFK	147	81	s	f	0x93	ESC [1 4 7 q	
81	Ctrl	PFK	148	81	c	f	0x94	ESC [1 4 8 q	
81	Alt	PFK	149	81	a	f	0x95	ESC [1 4 9 q	
82	Not available on keyboard								
83	Base	CUU	Cursor Up	83	b	f	0x101	ESC [A	
83	Shift	PFK	161	83	s	f	0xa1	ESC [1 6 1 q	
83	Ctrl	PFK	162	83	c	f	0xa2	ESC [1 6 2 q	
83	Alt	PFK	163	83	a	f	0xa3	ESC [1 6 3 q	
84	Base	CUD	Cursor Down	84	b	f	0x102	ESC [B	
84	Shift	PFK	164	84	s	f	0xa4	ESC [1 6 4 q	
84	Ctrl	PFK	165	84	c	f	0xa5	ESC [1 6 5 q	
84	Alt	PFK	166	84	a	f	0xa6	ESC [1 6 6 q	
85	Base	PFK	150	85	b	f	0x96	ESC [1 5 0 q	
85	Shift	PFK	151	85	s	f	0x97	ESC [1 5 1 q	
85	Ctrl	PFK	152	85	c	f	0x98	ESC [1 5 2 q	
85	Alt	PFK	153	85	a	f	0x99	ESC [1 5 3 q	
86	Base	PFK	154	86	b	f	0x9a	ESC [1 5 4 q	
86	Shift	PFK	155	86	s	f	0x9b	ESC [1 5 5 q	
86	Ctrl	PFK	156	86	c	f	0x9c	ESC [1 5 6 q	
86	Alt	PFK	157	86	a	f	0x9d	ESC [1 5 7 q	
87	Not available on keyboard								
88	Not available on keyboard								
89	Base	CUF	Cursor Forward	89	b	f	0x103	ESC [C	
89	Shift	PFK	167	89	s	f	0xa7	ESC [1 6 7 q	
89	Ctrl	PFK	168	89	c	f	0xa8	ESC [1 6 8 q	
89	Alt	PFK	169	89	a	f	0xa9	ESC [1 6 9 q	
90	Base	NUM LOCK		None				Not Returned	
90	Shift	NUM LOCK		None				Not Returned	
90	Ctrl	DC3	Device Control 3	90	c	c	< 19	0x13	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
90	Alt	PFK	170	90	a	f	0xaa	ESC [1 7 0 q	
91	Base	┌	Upper Left Corner	91	b	c	< 218	0xda	
91	Shift	7	Seven	91	s	c	< 55	0x37	
91	Ctrl	PFK	172	91	c	f	0xac	ESC [1 7 2 q	
91	Alt	Alt+Num Entry		None				Return at Alt Break	
92	Base	├	Left Edge Int.	92	b	c	< 195	0xc3	
92	Shift	4	Four	92	s	c	< 52	0x34	
92	Ctrl	PFK	174	92	c	f	0xae	ESC [1 7 4 q	
92	Alt	Alt+Num Entry		None				Return at Alt Break	
93	Base	└	Lower Left Corner	93	b	c	< 192	0xc0	
93	Shift	1	One	93	s	c	< 49	0x31	
93	Ctrl	PFK	176	93	c	f	0xb0	ESC [1 7 6 q	
93	Alt	Alt+Num Entry		None				Return at Alt Break	
94	Not available on keyboard								
95	Base	/	Slash	95	b	c	< 47	0x2f	
95	Shift	/	Slash	95	s	c	< 47	0x2f	
95	Ctrl	PFK	179	95	c	f	0xb3	ESC [1 7 9 q	
95	Alt	PFK	180	95	a	f	0xb4	ESC [1 8 0 q	
96	Base	┬	Top Intersection	96	b	c	< 194	0xc2	
96	Shift	8	Eight	96	s	c	< 56	0x38	
96	Ctrl	PFK	182	96	c	f	0xb6	ESC [1 8 2 q	
96	Alt	Alt+Num Entry		None				Return at Alt Break	
97	Base	┼	Center Intersection	97	b	c	< 197	0xc5	
97	Shift	5	Five	97	s	c	< 53	0x35	
97	Ctrl	PFK	184	97	c	f	0xb8	ESC [1 8 4 q	
97	Alt	Alt+Num Entry		None				Return at Alt Break	
98	Base	┴	Bottom Junction	98	b	c	< 193	0xc1	
98	Shift	2	Two	98	s	c	< 50	0x32	
98	Ctrl	PFK	186	98	c	f	0xba	ESC [1 8 6 q	
98	Alt	Alt+Num Entry		None				Return at Alt Break	
99	Base	|	Vertical Bar	99	b	c	< 179	0xb3	
99	Shift	0	Zero	99	s	c	< 48	0x30	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
99	Ctrl	PFK	178	99	c	f	0xb2	ESC [1 7 8 q	
99	Alt	Alt+Num Entry		None				Return at Alt Break	
100	Base	*	Asterisk	100	b	c	< 42	0x2a	
100	Shift	*	Asterisk	100	s	c	< 42	0x2a	
100	Ctrl	PFK	187	100	c	f	0xbb	ESC [1 8 7 q	
100	Alt	PFK	188	100	a	f	0xbc	ESC [1 8 8 q	
101	Base	┐	Upper Right Corner	101	b	c	< 191	0xbf	
101	Shift	9	Nine	101	s	c	< 57	0x39	
101	Ctrl	PFK	190	101	c	f	0xbe	ESC [1 9 0 q	
101	Alt	Alt+Num Entry		None				Return at Alt Break	
102	Base	┤	Right Edge Int.	102	b	c	< 180	0xb4	
102	Shift	6	Six	102	s	c	< 54	0x36	
102	Ctrl	PFK	192	102	c	f	0xc0	ESC [1 9 2 q	
102	Alt	Alt+Num Entry		None				Return at Alt Break	
103	Base	┘	Lower Right Corner	103	b	c	< 217	0xd9	
103	Shift	3	Three	103	s	c	< 51	0x33	
103	Ctrl	PFK	194	103	c	f	0xc2	ESC [1 9 4 q	
103	Alt	Alt+Num Entry		None				Return at Alt Break	
104	Base	─	Horizontal Line	104	b	c	< 196	0xc4	
104	Shift	.	Period	104	s	c	< 46	0x2e	
104	Ctrl	PFK	196	104	c	f	0xc4	ESC [1 9 6 q	
104	Alt	PFK	197	104	a	f	0xc5	ESC [1 9 7 q	
105	Base	-	Hyphen, Minus Sign	105	b	c	< 45	0x2d	
105	Shift	-	Hyphen, Minus Sign	105	s	c	< 45	0x2d	
105	Ctrl	PFK	198	105	c	f	0xc6	ESC [1 9 8 q	
105	Alt	PFK	199	105	a	f	0xc7	ESC [1 9 9 q	
106	Base	+	Plus Sign	106	b	c	< 43	0x2b	
106	Shift	+	Plus Sign	106	s	c	< 43	0x2b	
106	Ctrl	PFK	200	106	c	f	0xc8	ESC [2 0 0 q	
106	Alt	PFK	201	106	a	f	0xc9	ESC [2 0 1 q	
107	Not available on keyboard								

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
108	Base	CR	Carriage Return	108	b	c	< 13	0x0d	
108	Shift	CR	Carriage Return	108	s	c	< 13	0x0d	
108	Ctrl	CR	Carriage Return	108	c	c	< 13	0x0d	
108	Alt	PFK	100	108	a	f	0x64	ESC [1 0 0 q	
109	Not available on keyboard								
110	Base	ESC	Escape	110	b	c	< 27	0x1b	
110	Shift	PFK	120	110	s	f	0x78	ESC [1 2 0 q	
110	Ctrl	PFK	121	110	c	f	0x79	ESC [1 2 1 q	
110	Alt	PFK	122	110	a	f	0x7a	ESC [1 2 2 q	
111	Not available on keyboard								
112	Base	PFK	1	112	b	f	0x01	ESC [0 0 1 q	
112	Shift	PFK	13	112	s	f	0x0d	ESC [0 1 3 q	
112	Ctrl	PFK	25	112	c	f	0x19	ESC [0 2 5 q	
112	Alt	PFK	37	112	a	f	0x25	ESC [0 3 7 q	
113	Base	PFK	2	113	b	f	0x02	ESC [0 0 2 q	
113	Shift	PFK	14	113	s	f	0x0e	ESC [0 1 4 q	
113	Ctrl	PFK	26	113	c	f	0x1a	ESC [0 2 6 q	
113	Alt	PFK	38	113	a	f	0x26	ESC [0 3 8 q	
114	Base	PFK	3	114	b	f	0x03	ESC [0 0 3 q	
114	Shift	PFK	15	114	s	f	0x0f	ESC [0 1 5 q	
114	Ctrl	PFK	27	114	c	f	0x1b	ESC [0 2 7 q	
114	Alt	PFK	39	114	a	f	0x27	ESC [0 3 9 q	
115	Base	PFK	4	115	b	f	0x04	ESC [0 0 4 q	
115	Shift	PFK	16	115	s	f	0x10	ESC [0 1 6 q	
115	Ctrl	PFK	28	115	c	f	0x1c	ESC [0 2 8 q	
115	Alt	PFK	40	115	a	f	0x28	ESC [0 4 0 q	
116	Base	PFK	5	116	b	f	0x05	ESC [0 0 5 q	
116	Shift	PFK	17	116	s	f	0x11	ESC [0 1 7 q	
116	Ctrl	PFK	29	116	c	f	0x1d	ESC [0 2 9 q	
116	Alt	PFK	41	116	a	f	0x29	ESC [0 4 1 q	
117	Base	PFK	6	117	b	f	0x06	ESC [0 0 6 q	
117	Shift	PFK	18	117	s	f	0x12	ESC [0 1 8 q	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
117	Ctrl	PFK	30	117	c	f	0x1e	ESC [0 3 0 q	
117	Alt	PFK	42	117	a	f	0x2a	ESC [0 4 2 q	
118	Base	PFK	7	118	b	f	0x07	ESC [0 0 7 q	
118	Shift	PFK	19	118	s	f	0x13	ESC [0 1 9 q	
118	Ctrl	PFK	31	118	c	f	0x1f	ESC [0 3 1 q	
118	Alt	PFK	43	118	a	f	0x2b	ESC [0 4 3 q	
119	Base	PFK	8	119	b	f	0x08	ESC [0 0 8 q	
119	Shift	PFK	20	119	s	f	0x14	ESC [0 2 0 q	
119	Ctrl	PFK	32	119	c	f	0x20	ESC [0 3 2 q	
119	Alt	PFK	44	119	a	f	0x2c	ESC [0 4 4 q	
120	Base	PFK	9	120	b	f	0x09	ESC [0 0 9 q	
120	Shift	PFK	21	120	s	f	0x15	ESC [0 2 1 q	
120	Ctrl	PFK	33	120	c	f	0x21	ESC [0 3 3 q	
120	Alt	PFK	45	120	a	f	0x2d	ESC [0 4 5 q	
121	Base	PFK	10	121	b	f	0x0a	ESC [0 1 0 q	
121	Shift	PFK	22	121	s	f	0x16	ESC [0 2 2 q	
121	Ctrl	PFK	34	121	c	f	0x22	ESC [0 3 4 q	
121	Alt	PFK	46	121	a	f	0x2e	ESC [0 4 6 q	
122	Base	PFK	11	122	b	f	0x0b	ESC [0 1 1 q	
122	Shift	PFK	23	122	s	f	0x17	ESC [0 2 3 q	
122	Ctrl	PFK	35	122	c	f	0x23	ESC [0 3 5 q	
122	Alt	PFK	47	122	a	f	0x2f	ESC [0 4 7 q	
123	Base	PFK	12	123	b	f	0x0c	ESC [0 1 2 q	
123	Shift	PFK	24	123	s	f	0x18	ESC [0 2 4 q	
123	Ctrl	PFK	36	123	c	f	0x24	ESC [0 3 6 q	
123	Alt	PFK	48	123	a	f	0x30	ESC [0 4 8 q	
124	Base	PFK	209	124	b	f	0xd1	ESC [2 0 9 q	
124	Shift	PFK	210	124	s	f	0xd2	ESC [2 1 0 q	
124	Ctrl	PFK	211	124	c	f	0xd3	ESC [2 1 1 q	
124	Alt	PFK	212	124	a	f	0xd4	ESC [2 1 2 q	
125	Base	PFK	213	125	b	f	0xd5	ESC [2 1 3 q	
125	Shift	PFK	214	125	s	f	0xd6	ESC [2 1 4 q	
125	Ctrl	PFK	215	125	c	f	0xd7	ESC [2 1 5 q	

Key	Shift	Assignment		Keyboard Definition				Returned String	Notes
Posn	State			nnn	s	t			
125	Alt	PFK	216	125	a	f	0xd8	ESC [2 1 6 q	
126	Base	PFK	217	126	b	f	0xd9	ESC [2 1 7 q	
126	Shift	PFK	218	126	s	f	0xda	ESC [2 1 8 q	
126	Ctrl	DEL		126	c	c	< 127	0x7f	
126	Alt	DEL		126	a	c	< 127	0x7f	

Keystroke Control Sequences for System Functions

The following keystroke combinations cause the indicated system functions to be performed. The notation **Pad***n*, where *n* is a digit, indicates the *n* key on the numeric keypad to the right of the main keyboard area.

Note: Unless otherwise noted, the functions initiated by a three-key **Ctrl-Alt-***key* sequence require the **Alt** key on the *left* side of the standard RT PC keyboard. Functions initiated with **Alt-***key* (or **Shift-***key*) can be selected with either the left or the right **Alt** key (or **Shift** key).

AIX runs on a virtual machine, so all of the following references to "virtual machines" apply to the AIX Operating System.

AIX System Functions:

Alt-Pause Sends the interrupt signal, **SIGINT**, to all AIX processes associated with the terminal (or virtual terminal) from which this key sequence is entered. This causes most processes to terminate, although a process can arrange to ignore or take other action on this signal.

Ctrl-V Sends the quit signal, **SIGQUIT**, to all AIX processes associated with the terminal (or virtual terminal) from which this key sequence is entered. This causes most processes to terminate and produce a process image file in the current directory named **core**. However, a process can arrange to ignore or take other action on this signal.

See "termio" on page 6-114 for additional AIX keystroke control sequences.

Virtual Terminal Functions:

Alt-Action Changes the active display screen to the next virtual terminal (if any).

Shift-Action Changes the active display screen to the previous virtual terminal (if any).

Ctrl-Action Changes the active display screen to the command virtual terminal (if defined).

Coprocessor Functions:

These functions can be initiated with with either the left or the right **Alt** key.

Ctrl-Alt-Del Re-IPLs the coprocessor (if configured).

Ctrl-Alt-Action Exits the coprocessor direct mode.

IPL (System Restart) Functions:

Ctrl-Alt-Home Terminates all virtual machines and re-IPLs the virtual machines that are configured to be IPLed automatically. If diskette drive #1 contains a valid virtual machine IPL diskette, then only that virtual machine is IPLed.

Ctrl-Alt-Pause Performs a soft re-IPL, reloading the VRM as well as the virtual machines that are configured to be IPLed automatically.

Warning: This IPL function does *not* terminate the virtual machines before reloading the system. Data may be lost if you do not shut down all virtual machines before pressing this IPL key sequence.

Ctrl-Alt-Pad6 Performs a power-on reset re-IPL. This runs the Power-On Self Test (POST), then reloads the VRM and the virtual machines that are configured to be IPLed automatically.

Warning: This IPL function does *not* terminate the virtual machines before reloading the system. Data may be lost if you do not shut down all virtual machines before pressing this IPL key sequence.

IPL Device Specification Functions:

The following key sequences set a value in NVRAM that identifies the *second* device to be read during a VRM IPL. The system first checks diskette drive #1 for a valid IPL record, then the second device, then each of the fixed disks.

Ctrl-Alt-A Designates diskette drive #1 as the second device to be read.

Ctrl-Alt-B Designates diskette drive #2 (if configured) as the second devıce to be read.

Ctrl-Alt-C Designates fixed-disk drive #1 as the second device to be read.

Ctrl-Alt-D Designates fixed-disk drive #2 (if configured) as the second device to be read.

Ctrl-Alt-E Designates fixed-disk drive #3 (if configured) as the second device to be read.

System Dump Functions:

Note: Before attempting to use any of the following system dump key sequences, see *Software Problem Determination Guide* for more detailed information.

Ctrl-Alt-End	Performs a dump of the first virtual machine (usually AIX).
Ctrl-Alt-Pad8	Performs a system dump of selected parts of real memory (mostly VRM).
Ctrl-Alt-Pad7	Performs a dump of all real memory.

VRM Function:

Ctrl-Alt-Pad4	Invokes the VRM debugger.

Related Information

In this book: "data stream" on page 5-5, "display symbols" on page 5-24, "hft" on page 6-23, "Set Keyboard Map (HFSKBD)" on page 6-36, and "termio" on page 6-114.

Keyboard Description and Character Reference.

Software Problem Determination Guide.

lp

Purpose

Supports the line printer device driver.

Synopsis

#include <sys/lprio.h>

Description

The **lp** driver provides an interface to the port used by a printer. If an adapter for a printer is not installed, an attempt to open fails. The **close** system call waits until all output completes before returning to the user. The **lp** driver allows only one process to write to a printer adapter at a time. If the printer adapter is busy, the **open** system call returns an error. However, the driver allows multiple **open** system calls to occur if they are **read-only**. Thus, the **splp** command can be run when the printer adapter is currently in use.

The **lp** driver interprets carriage returns, backspaces, line feeds, tabs, and form feeds depending on the modes that are set in the driver (via **splp**). The number of lines per page, columns per line, and the indent at the beginning of each line can also be selected. The defaults are set at 66 lines per page, and 80 columns per line with no indenting.

ioctl Operations

Syntax for the enhanced control function is:

```
#include <sys/lprio.h>
ioctl (fildes,command,arg)
  int fildes;                       /* file descriptor */
  int command;                      /* command type */
  struct LPRUDE *arg;           /* pointer to info structure */
```

The possible command types and their descriptions are:

IOCINFO Returns a structure defined in **sys/devinfo.h**, which describes the device.

IOCTYPE Returns device LPR (line printer) defined in **sys/devinfo.h**.

LPRGET Gets page length, width, and indent. This structure is defined in **sys/lprio.h**.

LPRGETA Gets the RS232 parameters. These are the values for baud rate, character size, stop bits, and parity. Refer to the LPR232 structure and to the **termio.h** structure.

LPRGETV Gets optional line printer modes. See the following LPRMODE structure.

LPRGMOD Gets the RT PC optional printer modes. These optional printer modes support the synchronous versus asynchronous write interface, as well as the **report all errors** versus **wait until error correction** error reporting mode. Refer to the following **OPRMODE** structure.

The FONTINIT flag is initially off. It is turned on by an application when a printer font has been been initialized. It is turned off when an application wants fonts to be reinitialized and by the **lp** device driver when a FATAL printer error occurs.

LPRSET Sets page length, width, and indent values. This structure is defined in **sys/lprio.h**.

LPRSETA Sets the RS232 parameters, These are the values for baud rate, character size, stop bits, and parity. Refer to the LPR232 structure that follows and to the **termio.h** structure.

LPRSETV Sets optional line printer modes. See the following LPRMODE structure.

LPRSMOD Sets the RT PC optional printer modes. These optional printer modes support the synchronous versus asynchronous write interface, as well as the **report all errors** versus **wait until error correction** reporting mode. Refer to the following **OPRMODE** structure.

The FONTINIT flag is initially off. It is turned on by an application when a printer font has been been initialized. It is turned off when an application wants fonts to be reinitialized and by the **lp** device driver when a FATAL printer error occurs.

LPRUFLS Flushes any data currently in progress and causes the printer to be initialized. Due to VRM queuing mechanism, printing of the data in the current queue element may take some time to complete before printing stops. This can be used during the course of normal print operations, or following an error indication.

LPRUGES Gets the device driver error structure (LPRUDE). The *arg* parameter must be specified to point to the structure.

LPRURES Resumes printing from the point of interruption following an error. If an error did not occur, then this control has no effect.

LPRVRMS Sets VRM device-dependent parameters. This causes a set device characteristics Start–I/O SVC to be issued with the device-dependent parameters.

LPRVRMG Gets Virtual Resource Manager (VRM) device-dependent parameters. This returns the structure obtained by issuing a Query_Device SVC. The structure contains status information, hardware characteristics, device-dependent parameters, and RAS log information.

LPRGTOV Gets the current time-out value and stores it in the **lptimer** structure pointed to by the *arg* parameter. The time-out value is measured in seconds.

LPRSTOV Sets the time-out value. The *arg* parameter points to a **lptimer** structure. The time-out value must be given in seconds.

Most of these **ioctl** operations require the *arg* parameter to point to one of the following structures:

```
struct devinfo
{ char devtype; /* devtype for printer is '1' */
  char flags;
};

/* used with LPRGET, LPRSET */
struct LPRIO {
   int  ind;    /* indent value */
   int  col;    /* maximum character count */
   int  line;   /* maximum line count */
};

/* used with LPRGET, LPRSET */
struct LPRMODE {
   int  modes;  /* optional line printer modes */
};
/* bit definitions for the modes field in LPRMODE */
#define PLOT  01      /* if on, no interpretation of any character */
#define NOFF  0400    /* if on, simulate the form-feed function */
#define NONL  01000   /* if on, substitute carriage returns for */
                      /* any line-feeds */
#define NOTAB 02000   /* if on, don't expand tabs, else simulate */
                      /* 8 position tabs with spaces */
#define NOBS  04000   /* if on, no backspaces to the printer */
#define NOCR  010000  /* if on, substitute line-feeds for any   */
                      /* carriage returns */
#define CAPS  020000  /* if on, map lower-case alphabetics */
```

```
                        /* to upper case */
#define WRAP  040000  /* if on, print characters beyond the page */
                        /* width on the next line, instead of */
                        /* truncating */

struct OPRMODE {
  int  flags;            /* optional line printer modes */
};
#define SYNC    01       /* asynchronous is default. */
                         /* synchronous if on */
#define ALLERR  02       /* wait until error correction is default.*/
                         /* report all errors if on */
#define FONTINIT 04      /* file initialization */
struct LPRUDE            /* device error-reporting structure */
{
  int  status;           /* error reason code */
  int  cresult;          /* current operation result :PSB */
  int  tadapt;           /* adapter type */
  int  npio;             /* number of pending IO operations */
};

/* status values - error reason codes */

struct LPR232            /* settings for RS232 */
{
unsigned c_cflag;        /* error reason code */
};

/* used with LPRGTOV and LPRSTOV */
struct lptimer
{
   unsigned v_timeout; /* time-out value in seconds */
}
```

Files

/dev/lp*

Related Information

In this book: "ioctl" on page 2-56 and "devinfo" on page 4-57.

The **splp** command in *AIX Operating System Commands Reference.*

mem, kmem, nvram

Purpose

Provides memory, kernel memory, and non-volatile memory images.

Description

The **mem, kmem**, and **nvram** files are pseudo-device driver files. These device drivers are an image of physical memory in the system. These files can be used, for example, to examine or to patch the system.

The **mem** file is a special file that is an image of the system-generated virtual memory. It can be used, for example, to examine, and even to patch the system. You should use the **readx** and **writex** system calls to read or write **mem**. The parameter is the segment ID of the memory segment to be read or written. The system uses the seek offset as the byte offset within the segment. The high-order 4 bits of the seek offset are ignored. If there is a segment ID of 0, or if you use **read** instead of **readx, mem** acts like **kmem**.

The file **kmem** is similar to **mem**, except it is used to access kernel and calling program virtual memory. The seek offset is an address in kernel virtual memory. Seek offsets of less than 0x1000 0000 address the kernel segment, which contains both text and data. Seek offsets from 0x1000 0000 to 0x1FFF FFFF address the text segment of current process and so on.

The **nvram** file is used to access the system non-volatile memory. It contains an area 16 bytes long to log RT PC errors when the system cannot make a permanent copy of errors on the disk. Unlike **mem** and **kmem**, information written to this device is retained after power is removed from the system.

An invalid virtual address or segment ID used with **mem, kmem**, or **nvram** causes errors to be returned.

Files

/dev/mem
/dev/kmem
/dev/nvram

null

Purpose

Provides a null device.

Description

The **null** file is a pseudo-device driver file with no associated hardware. Data written to this file is discarded. Reads from this file always return 0 bytes. Use this file to read or write null data as required.

File

/dev/null

osm

Purpose

Provides the interface to AIX messages.

Description

The **osm** driver collects system messages provided by the AIX kernel and application programs. These system messages are available to a daemon reading this file. System messages have two sources:

- The AIX kernel provides messages by calls to the kernel **printf** routine.
- Application programs open and write to this file.

Operating system messages are stored in a circular buffer in the system and can be read or written using the **osm*** special files. A read from **osm*** files returns some portion of the data in the circular buffer. A write to the files adds user data to the current end of the circular buffer. Any number of users may use **osm*** files in the same instance of time.

Read operations from the **osm** file start at the current end of the circular buffer and wait for new data to be added. Read operations from the file **/dev/osm.curr** start at the beginning of the circular buffer and return 0 bytes when the current end of the buffer is reached. Read operations from the **/dev/osm.all** file start at the beginning of the circular buffer, go to the current end of the circular buffer, and wait for new data to be added.

Files

/dev/osm*

Related Information

In this book: "rasconf" on page 4-133.

prf

Purpose

Profiles the kernel.

Description

The **prf** file is a pseudo-device driver file with no associated hardware. This device driver file provides access to activity information in the operating system.

This file is initialized to contain an array of sorted kernel text addresses. An **ioctl** system call issued with command value 3 and parameter value equal to 1 starts monitoring operations. Each subsequent clock interrupt causes the table of addresses to be searched and the highest address less than or equal to the program counter at the time of the interrupt to be located. The counter corresponding to the located address is incremented. An interrupt that occurs while in user mode, increments a miscellaneous counter. An **ioctl** system call issued with command value 3 and parameter value equal to 0 stops the monitoring.

Reading this file returns the array of addresses and the array of counters. The miscellaneous counter is returned last. The information is returned in a single **read** system call. The buffer supplied must be large enough to hold the information.

You can determine the status of the profiling facility, by issuing an **ioctl** system call with command value 1. Bit 1 (the least significant bit) of the return value is set if monitoring is on. Bit 2 bit is set if a valid list of text addresses was loaded. An **ioctl** system call issued with a command value 2 returns the number of loaded text addresses.

File

/dev/prf

Related Information

The **config** and **profiler** commands in *AIX Operating System Commands Reference.*

pty

Purpose

Implements a pseudo-terminal device.

Synopsis

#include <sys/devinfo.h>
#include <sys/pty.h>
#include <sys/tty.h>

Description

A **pty** device is a pair of bi-directional character device drivers that implement a pseudo-terminal. A pseudo-terminal can act as a keyboard and a display to existing software that uses the standard terminal device interface described in "termio" on page 6-114. This is useful for a variety of applications such as a remote login facility or a windowing system.

Each pseudo-terminal (or **pty**) consists of two device drivers called a controller and a server. The ***server*** or ***server side*** of a **pty** has a standard terminal interface that can support a login shell or other software that normally communicates with terminals. The ***controller*** or ***controller side*** of a **pty** interfaces with software that generates and receives data as if it were a user at a terminal. Data written to the controller is passed directly to the server, which is then read and processed as if entered from a keyboard. Data written to the server (as if to be displayed on a terminal screen) is passed directly to the controller.

The corresponding special files are named **/dev/ptc***n* for the controller and **/dev/pts***n* for the server, where *n* is a decimal number. A 1-to-1 correspondence exists between each controller-server pair with names that end in the same number. For example, **/dev/ptc0** and **/dev/pts0** together form a **pty**.

The **ptybuffers** keyword in the **/etc/master** file controls the number of **pty**s that can be present in the system. The maximum number of **pty**s is 16.

For a remote login application such as Telnet, use the **devices** command to add the server side of a **pty** to the system configuration and enable it as a login port. In the case of Telnet, the **telnetd** daemon opens the controller side of the **pty** and passes data sent over the network to the login shell.

When using a **pty** for applications other than remote login, a program must take into account the fact that a logger process may have already issued an **open** to the server side of the **pty**. When a logger opens the server side, the **open** system call suspends the process to wait for another process to open the controller side. Use the following strategy to detect this situation:

1. Open **/dev/ptc***n*.
2. Issue an **ioctl** system call to perform the **PTYSTATUS** operation.
3. If the status indicates that the server side has already been opened, then close the **pty** controller and try a **/dev/ptc***n* device with a different value for *n*.
4. If the status indicates that the server side has not been opened, then open the corresponding **/dev/pts***n* device.

Note: The server side of a **pty** can be opened multiple times, but the controller can be opened only once. Attempting to open the controller side more than once causes an error.

select Support

The **pty** device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are satisfied when data can be accepted.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See "select" on page 2-111 for more information about this system call.

ioctl Operations

The interface to the server side of the **pty** device is identical to the standard interface for terminals, which is described in "termio" on page 6-114.

The controller side of the **pty** device driver supports the following **ioctl** operations. (See "ioctl" on page 2-56 for detailed information about the **ioctl** system call.)

IOCTYPE Returns the device type **DD_PSEU** to indicate that this is a pseudo-terminal device. This operation ignores the *arg* parameter.

IOCINFO Copies the **devinfo** structure for the device into the buffer pointed to by the *arg* parameter passed to **ioctl**. See "devinfo" on page 4-57 for details about this structure.

PTYSTATUS Returns the state of the **pty**, which is composed of two half-words. The upper half contains the number of opens currently outstanding against the controller, and the lower half contains the number of opens currently outstanding against the server. This operation ignores the *arg* parameter.

PTYIOR Reports the number of characters available to be read. The *arg* parameter is a pointer to an integer, into which this value is stored.

PTYIOW Reports the number of characters on the raw and cannonical queues. The *arg* parameter is a pointer to an integer, into which this value is stored.

PTYGETM Gets the current mode of the **pty**. The *arg* parameter is a pointer to an integer, into which the mode is stored. See the description **PTYSETM** for an explanation of the possible mode values.

PTYSETM Sets the current mode of the **pty**. The *arg* parameter is a pointer to an integer that contains the mode to be set. The mode is zero or more of the following values logically OR-ed together:

- **RAWQINT** Sends the **SIGPTY** signal to the process when enough buffer space is available for writing to the **pty**.
- **OUTQINT** Sends the **SIGPTY** signal to the process when data is available to be read.
- **REMOTE** Controls the flow of input to the **pty**, but does not edit the input. In other words, START and STOP (**Ctrl-S** and **Ctrl-Q**) controls are processed, but no editing is done on the data stream (such as ERASE, KILL, or ICRNL).

PTYADDM Adds to the current **pty** mode by logically OR-ing the specified value with the existing mode. The *arg* parameter is a pointer to an integer that contains the mode bits to be set. See the description **PTYSETM** for an explanation of the possible mode values.

PTYDELM Deletes from the current **pty** mode. The *arg* parameter is a pointer to an integer. The bits that are set in this integer specify the mode bits to be turned off. See the description **PTYSETM** for an explanation of the possible mode values.

Diagnostics

System calls to a **pty** device fail and set **errno** to indicate the error if one or more of the following are true:

EINVAL An invalid parameter was encountered, such as a negative number of bytes to be written.

ENXIO The **pty** cannot be opened because the **pty** number is out of range.

EIO A **read**, **write**, or **ioctl** operation was attempted that requires both sides of the **pty** to be open, making a complete connection.

EACCES An attempt was made to open the controller side of a **pty** more than once.

Files

/dev/ptc0, /dev/ptc1, . . .	Controller Devices
/dev/pts0, /dev/pts1, . . .	Server Devices

Related Information

In this book: "ioctl" on page 2-56, "open" on page 2-90, "master" on page 4-98, "ports" on page 4-117, "system" on page 4-139, and "fcntl.h" on page 5-56.

The **tn** and **telnetd** commands in *Interface Program for use with TCP/IP.*

The **devices** and **init** commands in *AIX Operating System Commands Reference.*

tape

Purpose

Supports the sequential access bulk storage medium device driver.

Description

Magnetic tapes are used primarily for backups, file archives, and other off-line storage. Tapes are accessed through the special files **rmt0, . . . , rmt15**. The **r** indicates "raw" which indicates access through the character special interface. A streaming tape does not lend itself well to the category of a block device; only these character interface files are provided. The number following the **rmt** is the minor device number. The two low-order bits of the minor device number select the transport. If the third bit (04 octal or 0x04) is set, the driver does not rewind the tape after it is closed. If the fourth bit (010 octal or 0x08) is set, the tape is retensioned (wound completely forward and then rewound) after it is opened and before any other operations.

On a system with a single tape drive, **/dev/rmt0** does not retension the tape, but does rewind it on close. **/dev/rmt4** (bits = 0100) does not perform any special actions on open or close. **/dev/rmt8** (bits = 1000) retensions the tape and rewinds it on close; and **/dev/rmt12** (bits = 1100) retensions the tape on open, but does not rewind.

When opened for reading or writing, the tape is assumed to be positioned as desired. When the tape opens and writes to a file, a single tape mark is written if the file is no rewind on close, while a double tape mark is written if the tape is to be rewound. If the file is no rewind and opened read only, the tape is positioned after the end of file (EOF) following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF. By specifically choosing **rmt** files, it is possible to read and write multiple-file tapes.

Each **read** or **write** call reads or writes the next record on the tape. The record written by **write** is the same length as the buffer given. During a **read**, the record size is returned as the number of bytes read, up to the buffer size specified. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned before the EOF.

A number of **ioctl** operations are available. In addition to IOCTYPE and IOCINFO types, the following **ioctl** calls are defined.

The parameter to the **ioctl** system call using the STIOCTOP command is the address of a **stop** structure, which contains the following members:

```
short   st_op;      /* Streaming tape operation */
daddr_t st_count;   /* Number of times to perform */
```

The **st_op** operation is performed **st_count** times, except where it is not logical to do so, rewind, as an example. The operations available are:

```
#define STRESET 5   /* reset device */
#define STREW 6     /* rewind */
#define STERASE 7   /* erase tape, retension, leave at load point */
#define STRETEN 8   /* erase tape, retension, leave at load point */
#define STWEOF 10   /* write an end-of-file record */
#define STFSF  11   /* forward space file */
#define STFSR  13   /* forward space record */
#define STRAS1 15   /* drive self test 1 */
#define STRAS2 16   /* drive self test 2 */
#define STRAS3 17   /* drive self test 3 */
                   /* this test needs an */
                   /* erased write-protected tape */
```

The status of a tape drive can be determined by issuing the following STIOCGET type **ioctl** system call:

```
/* structure for STIOCGET - streaming tape get status command */
struct  stget  {
   short  st_type;      /* type of device */
  struct dsreg {
          unsigned short ds_dstat:  /* drive status */
          unsigned short ds_soft;   /* soft error count */
          unsigned short ds_under;  /* underrun count */
          unsigned char  ds_rcom;   /* command received by adapter */
          unsigned char  ds_blk;    /* adapter block count */
          unsigned char  ds_rstat;  /* status register */
          unsigned char  ds_code;   /* adapter completion code */
          unsigned char  ds_lcom;   /* last command given to adapter */
          unsigned char  ds_lstcom; /* last streaming tape device */
                                   /* drive command */
          unsigned char  ds_res[4]  /* reserved */
  } st_dsreg;
};

/*
* Constants for st_type byte - ST_SST streaming tape
*/
```

In addition to those errors listed in **ioctl**; **open**, **read**, and **write**, system calls against this device fail in the following circumstances:

EINVAL O_APPEND is supplied as a mode in which to open.

EINVAL A write attempt while the tape is in read mode, or a read attempt while the tape is in write mode.

EINVAL A **count** parameter to **read** or **write** is not 0, modulo 512.

EIO A parameter to **ioctl** is not allowed in the current streaming mode.

ENXIO The tape is write-protected or there is no tape in the drive.

Note: The streaming tape device driver has a concept of current "streaming mode." Therefore, many operations are invalid most of the time. In particular, no reads are allowed after an initial write or writes allowed after an initial read. You must wait until the device is reset either by closing a rewind-on-close special file, or by the **tctl** command.

Files

/dev/rmt*

Related Information

In this book: "ioctl" on page 2-56, "open" on page 2-90, "read, readx" on page 2-106, and "write, writex" on page 2-184.

The **tctl** command in *AIX Operating System Commands Reference.*

termio

Purpose

Provides the general terminal interface.

Synopsis

#include <sys/hft.h>
#include <sys/termio.h>
#include <sys/tty.h>

Description

All of the asynchronous communications ports use the same general interface, regardless of the hardware used. This discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, user programs seldom open these files. They are opened by **getty** and become standard input, output, and error files for a user. The first terminal file not already associated with a process group that is opened by the process group leader becomes the **control terminal** for that process group. The control terminal plays a special role in handling quit and interrupt signals as discussed later. During a **fork** system call, the child process inherits the control terminal. A process can break the association to the group using the **setpgrp** system call.

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters can be typed at any time, even while output is occurring. These characters can be lost, however, when the input buffers become completely full or when the user accumulates the maximum number of input characters allowed that were not read by a program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are erased from the input buffer without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read is suspended until an entire line is typed. Also, no matter how many characters are requested in the read call, at most one line is returned. It is not, however, necessary to read a whole line at once. Any number of characters can be requested in a read without losing information.

During input, erase and kill processing is performed normally. By default, the **Ctrl-H** character erases the last character typed, but does not erase beyond the beginning of the line. By default, the **Ctrl-U** character "kills" (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that was done. Both the erase and kill characters can be entered literally by preceding them with the \ (backslash) escape character. In this case, the escape character is not read. The erase and kill characters can be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

EOF **Ctrl-D** or ASCII EOT is used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line character, and the EOF is discarded. Thus, if there are not any characters waiting (indicating the EOF occurred at the beginning of a line), zero characters are passed back, which is the standard end-of-file indication.

EOL ASCII NUL is an additional line delimiter, like NL. It is not normally used.

ERASE **Ctrl-H** erases the preceding character. It does not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character.

INTR **Rubout** or ASCII DEL (**Ctrl-Backspace** on the RT PC console keyboard) generates a **SIGINT** (interrupt) signal, which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements can be made either to ignore the signal or to receive a trap to an agreed-upon location. See "signal" on page 2-145.

KILL **Ctrl-U** deletes the entire line, as delimited by an NL, EOF, or EOL character.

NL ASCII LF is the normal line delimiter. It cannot be changed or escaped.

QUIT **Ctrl-V** or ACSII SYN generates a **quit** signal. Its treatment is identical to the interrupt signal except that, unless a receiving process made other arrangements, it is not only terminated but a memory file (called **core**) is created in the current working directory.

START **Ctrl-Q** or ASCII DC1 is used to resume output that was suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters cannot be changed or escaped.

STOP **Ctrl-S** or ASCII DC3 is used to temporarily suspend output. It is useful with terminals that have displays to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL can be changed to suit individual preferences. The ERASE, KILL, and EOF characters can be escaped by a preceding \ (backslash) character, in which case the special function is not done.

When the carrier signal from the dataset drops, a **hangup** signal (SIGHUP) is sent to all processes that have this terminal as the control terminal. Unless other arrangements were made, this signal causes the process to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately.

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters finish typing. Input characters are usually echoed by putting them in the output queue as they arrive, but see "Enhanced Edit Mode" on page 6-122. If a process produces characters more rapidly than they can be typed, it is suspended when its output queue exceeds some limit. When the output decreases to a determined threshold, the program is resumed.

Several **ioctl** system calls apply to terminal files. The primary calls use the following structures defined in the **termio.h** header file:

```
#define NCC  8
struct termio  {
     unsigned  short  c_iflag;  /* input modes */
     unsigned  short  c_oflag;  /* output modes */
     unsigned  short  c_cflag;  /* control modes */
     unsigned  short  c_lflag;  /* local modes */
     char             c_line;   /* line discipline */
     unsigned  char   c_cc[NCC];  /* control chars */
};

struct  tty_page  {
    char  tp_flags;
    unsigned char  tp_slen;
};
```

The special control characters are defined by the **c_cc** array. The relative positions and initial values for each function are as follows:

c_cc[0]	INTR	**Ctrl-Backspace** (DEL)
c_cc[1]	QUIT	**Ctrl-V** (SYN)
c_cc[2]	ERASE	**Backspace** (BS)
c_cc[3]	KILL	**Ctrl-U** (NAK)
c_cc[4]	EOF	**Ctrl-D** (EOT)
c_cc[5]	EOL	**Ctrl-@** (NUL)
c_cc[6]	reserved	
c_cc[7]	reserved	

The **c_iflag** field describes the basic terminal input control. The initial input control value is all bits clear. The possible values are:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map new-line character (NL) to carriage return character (CR) on input.
IGNCR	0000200	Ignore carriage return character.
ICRNL	0000400	Map carriage return character to new-line character on input.
IUCLC	0001000	Maps uppercase to lowercase on input.
IXON	0002000	Enables start/stop output control.
IXANY	0004000	Enables any character to restart output.
IXOFF	0010000	Enables start/stop input control.
ASCEDIT	0020000	Enables enhanced editing on ASCII terminals.

The values in this field are described as follows:

IGNBRK If set, the break condition (a character framing error with data all zeros) is ignored. It is not put on the input queue and therefore not read by any process. Otherwise, if BRKINT is set, the break condition generates an interrupt signal and flushes both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

PARMRK If set, a character with a framing or parity error that is not ignored is read as the 3-character sequence: 0377, 0, *x*, where *x* is the data of the character received in error. If ISTRIP is not set, then a valid character of 0377 is read as 0377, 0377 to avoid ambiguity. If PARMRK is not set, a framing or parity error that is not ignored is read as the character NUL (0).

INPCK If set, input parity checking is enabled. If not set, input parity checking is disabled. This allows output parity generation without input parity errors.

ISTRIP If set, valid input characters are first stripped to 7 bits; otherwise all 8 bits are processed.

INLCR If set, a received new-line character is translated into a carriage-return character. If IGNCR is set, a received carriage-return character is ignored (not read). If ICRNL is set, a received carriage-return character is translated into a new-line character.

IUCLC If set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

IXON If set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. All start/stop characters are ignored and not read. If IXANY is set, any input character restarts output that was suspended.

IXOFF If set, the system transmits START/STOP characters when the input queue is nearly empty or full.

ASCEDIT If set, ASCII keyboards can be used to enter enhanced edit line discipline commands.

The **c_oflag** field specifies how the system treats output. The initial output control value is all bits clear.

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lowercase to uppercase on output.
ONLCR	0000004	Map new-line character to CR-NL on output.
OCRNL	0000010	Map carriage-return to new-line on output.
ONOCR	0000020	No carriage-return character output at column 0.
ONLRET	0000040	Perform carriage return function using new-line character.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL or NUL.
NLDLY	0000400	Select new-line character delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

OPOST If set, output characters are post-processed as indicated by the remaining flags; otherwise characters are transmitted without change.

OLCUC If set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

ONLCR If set, the new-line character is transmitted as the carriage-return new-line character pair.

OCRNL If set, the carriage-return character is transmitted as the new-line character.

ONOCR If set, no carriage-return character is transmitted when at column 0 (first position).

ONLRET If set, the new-line character is assumed to do the carriage return function. The column pointer is set to 0 and the delay specified for carriage return is used. Otherwise the new-line character is assumed to do just the line feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the carriage-return character is actually transmitted.

OFILL If set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay.

OFDEL If set, the fill character is DEL, otherwise NUL.

NLDLY, CRDLY, TABDLY, BSDLY, VTDLY, FFDLY
The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If ONLRET is set, the carriage return delays are used instead of the new-line delays.

TAB3 If set, specifies that tabs are to be expanded into spaces.

The **c_cflag** field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A
EXTB	0000017	External B

CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send 2 stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

CBAUD These bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal is not dropped. Normally, this disconnects the line. For any particular hardware, impossible speed changes are ignored.

CSIZE These bits specify the character size in bits for both transmit and receive. This size does not include the parity bit, if any. If CSTOPB is set, 2 stop bits are used; otherwise one stop bit is used. For example, at 110 baud, 2 stop bits are required.

CREAD If set, the receiver is enabled. Otherwise characters are not received.

PARENB If set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set; otherwise even parity is used.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

HUPCL If set, the line is disconnected when the last process that has the line open, either closes it or the process terminates. That is, the data-terminal-ready signal drops.

CLOCAL If set, the line is assumed to be local, direct connection with no modem control. Otherwise modem control is assumed.

The **c_lflag** field of the parameter structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.

ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo new-line character after kill character.
ECHONL	0000100	Echo new-line character.
NOFLSH	0000200	Disable flushing the queue after interrupt or quit.

ISIG If set, each input character is checked against the special control characters INTR and QUIT. If a character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, checking is not done. Thus, these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (for example, 0377 octal or 0xFF).

ICANON If set, canonical processing is enabled. Canonical processing enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, then read requests are satisfied directly from the input queue. In this case, a read request is not satisfied until either at least MIN characters have been received, or the time-out value TIME has expired since the last character was received. This allows bursts of input to be read, while still allowing single-character input. The MIN and TIME values are stored in the positions for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

XCASE If set along with ICANON, an uppercase letter (or the uppercase letter translated to lowercase by IUCLC) is accepted on input by preceding it with a \ (backslash) character, and is output preceded by a \ (backslash) character. In this mode, the output generates and the input accepts the following escape sequences:

<table>
<tr><th>For:</th><th>Use:</th></tr>
<tr><td>`</td><td>\'</td></tr>
<tr><td>|</td><td>\!</td></tr>
<tr><td>~</td><td>\^</td></tr>
<tr><td>{</td><td>\(</td></tr>
<tr><td>}</td><td>\)</td></tr>
<tr><td>\</td><td>\\</td></tr>
</table>

For example, A is input as \a, \n as \\n, and \N as \\\n.

ECHO If set, characters are echoed as received. When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which clears the last character from a cathode-ray-tube screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the new-line character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the new-line character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (sometimes called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

NOFLSH If set, the normal flushing of the input and output queues associated with the quit and interrupt characters is not done.

Enhanced Edit Mode

The **c_line** field describes the line-discipline control value. The initial line-discipline control value is all bits clear. When **c_line** is equal to 1, it sets enhanced edit mode. This terminal line discipline provides a simple, line-oriented editing facility modeled on DOS Services.

The enhanced edit line discipline supports the same flags in **c_lflag** as the basic line discipline, but has the differences described following. The line discipline itself may be used from any terminal. To set the terminal to this mode from the shell, use the **stty** command. From a program, use the **ioctl** system call.

In enhanced edit mode, a special character buffer called the ***template*** is associated with the terminal. Using special function keys, the next line entered can be constructed out of the template and new characters.

Initially the template is blank. When a line is read by a running program, that line becomes the active template. The template can also be explicitly read and set by the application program using the **ioctl** system call and by the user with the F5 key. The template does not directly appear on the terminal, but can be inspected by use of the function keys. When **Enter** is pressed, the old template is stacked and the current line becomes the active template. Up to eight templates can be stacked. The oldest template is deleted when a template is added to a full stack. The ↑ (**cursor up**) and ↓ (**cursor down**) keys change the active template from current template to the next or previous template respectively. The user can scroll through the stack, which automatically wraps back to the beginning when the end is reached.

Characters typed are not echoed to the terminal until the application program has issued a **read** system call to process it. The ERASE character is echoed as follows: when an ASCII TAB is deleted, the cursor is moved to the position where the TAB was typed, and the cursor will not be moved to the left of where input began on the current line.

Whenever non-printing characters are directly echoed to the terminal, they appear as ^*X*, where *X* is the printable character that is 64 greater in value than the non-printing character originally entered. Thus **Ctrl-A** is echoed as ^A, and so on. Finally, because DOS Services itself lacks any convention for escaping special characters, there is no way to specify literal occurrences of ERASE, KILL, or EOF. The \ (backslash) has no special meaning when used preceding them, and no escape character is defined.

Both the line buffer and the template can hold at least 128 characters. If the line buffer fills up, an audible signal sounds upon receipt of further characters, which are otherwise ignored. Exceptions are the **Backspace** and ← keys, which delete characters from the line buffer; and the **Esc**, **F5**, and **Return** (or **Enter**) keys, which reset the line buffer after performing their functions.

While this mode is intended primarily for use with the **DOS Services** commands, it can be set at any time, although it may be overridden by programs such as editors that change the terminal characteristics in other ways.

To understand the use of the special functions keys, it is helpful to consider their effect on the current line buffer, the template index, which points to a character in the template.

To use enhanced edit mode from an ordinary ASCII terminal, a compatibility mode must be set.

Native	**ASCII**	
Keyboard	**Keyboard**	**Action**
Displayable Character	Displayable Character	Places the character typed in the line buffer. Advances the template buffer unless insert mode is on. (When insert mode is on, characters typed are effectively "inserted" into the line within the template. When off, characters typed effectively "overwrite" characters in the template.)
Return or **Enter**	**Return** or **Enter**	Sends the line buffer (as it appears on the screen) to the application program and accepts it as the active template. The line on the screen is advanced and the old template is placed on the stack. The line buffer is emptied and the template index reset to the beginning of the template.
Esc	**Esc Esc**	Cancels the line currently being edited. A \ (backslash) character is echoed, then the cursor is moved to the same column on the next line as it was on the current line. The line buffer is emptied, and the template index is reset to the beginning of the template.
Del	**Esc D**	Advances the template index by one. There is no effect on the screen. This, in effect, deletes the next template character, although **Backspace** restores it.
↑	Esc H	Changes the active template to the next one on the stack and copies the new active template to the line buffer. Advances the template index to the end of the template.

Native Keyboard	ASCII Keyboard	Action
Ins	**Esc I**	Sets insert mode on. Insert mode is reset by ↑, ↓, **F1**, **F2**, **F3**, **Esc**, and **Enter**. When insert mode is set, inserts typed characters into the line within the template. When not set, characters that are typed overwrite characters in the template.
Backspace or ←	**Backspace** or **Esc J**	Backspaces and removes a character from the screen, moving the template index back one, but not beyond the column where text entry began. If insert mode is not set, move the template index back but beyond the beginning of the template. Backspacing across a tab character moves the cursor to the position of the character before the tab character was typed.
↓	**Esc L**	Changes the active template to the previous one in the stack and copies the new active template to the line buffer. Advances the template index to the end of the template.
F1 or →	**Esc K** or **Esc 1**	Copies the character indicated by the template index to the line buffer and displays it. Advances the template index.
F2	**Esc 2**	Copies characters (as **F1** does) up to, but not including, the next character typed.
F3	**Esc 3**	Copies to the line buffer, the characters in the template from the template index to the end of the template. Advances the template index to the end of the template.
F4	**Esc 4**	Advances the template index without copying characters (like **Del**) up to the next character typed. **F4** is to **Del** what **F2** is to **F1**.
F5	**Esc 5**	Accepts the current line buffer as a new template. The @ character is echoed, and then the cursor is moved to the next line at the same column as that at which it started on the current line. The template index is reset to the beginning of the template.
F6	**Esc 6**	Enters a **Ctrl-Z** character into the line buffer, as if it had been typed directly from the keyboard.
F7	**Esc 7**	Enters an ASCII NUL character into the line buffer, as if it had been typed directly from the keyboard.

Any other character typed is placed in the line buffer, except when ICANON is set. When ICANON is set, the special characters it provides are not retained in the buffer. The template index is advanced whenever a character is placed in the line buffer, unless insert mode is enabled.

The system console has other specific modes that are not valid for general terminal interfaces. See “hft” on page 6-23 for details.

select Support

The asynchronous terminal device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are always satisfied immediately.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See “select” on page 2-111 for more information about this system call.

ioctl Operations

The primary **ioctl** system calls have the format:

```
ioctl (fildes, command, arg)
int fildes;        /* file descriptor */
int command;       /* command type */
struct termio *arg;
```

The commands using this format are:

TCGETA Gets the parameters associated with the terminal and stores them in the **termio** structure referenced by *arg*.

TCSETA Sets the parameters associated with the terminal from the structure referenced by *arg*. The change is immediate.

Note: **TCGETA** and **TCSETA** do *not* get and set a complete record of the state of an HFT device. See “hft” on page 6-23 for information about high-function terminal devices.

TCSETAF Waits for the output to empty, then flushes the input queue and sets the new parameters.

TCSETAW Waits for the output to empty before setting the new parameters. This form should be used when changing parameters that affect output.

The terminal paging **ioctl** calls have the format:

```
ioctl (fildes, command, arg)
int fildes;      /* file descriptor */
int command;     /* command type */
struct tty_page *arg;
```

The commands using this format are:

TCGLEN Gets the current status of the **tty_page** structure for the terminal specified as **fildes**. If paging is enabled, a value 0x1 is set in **tp_flags**. The **tp_slen** value indicates the screen length in lines.

TCSLEN Sets the status of the **tty_page** structure for this terminal. **tp_slen** means the same here as it does in TCGLEN. The **tp_flags** are:

PAGE_SETL	0x4	Set page length using the value in **tp_slen**.
PAGE_MSK	0x3	Command mask.
PAGE_ON	0x1	Enable paging.
PAGE_OFF	0x2	Disable paging.

Note that the PAGE_MSK field is interpreted as an encoding, not as separate flags.

Additional **ioctl** system calls formats are:

```
ioctl (fildes, command, arg)
int fildes;     /* file descriptor */
int command;    /* command type */
int arg;
```

The commands using this format are:

TCFLSH If *arg* is 0, flush the input queue. A value of 1 indicates flush the output queue. A value of 2 indicates flush both the input and output queues.

TCSBRK Waits for the output to empty. If *arg* is 0, then sends a break (zero bits for 0.25 seconds).

TCXONC Starts or stops control. Suspends output if *arg* is 0. Restarts suspended output if *arg* is a value of 1.

One query **ioctl** system call has the following format:

```
ioctl (fildes, command, &arg)
int arg;     /* returned value */
```

The call using this format is:

TIONREAD Gets the summation of the number of characters in the raw and canonical queues.

Two **ioctl** system calls specific to the enhanced edit line discipline have the format:

```
ioctl (fildes, command, arg)
struct dostmplt *arg;
```

The **dostmplt** structure is defined in the **sys/termio.h** header file, and it contains the following members:

```
char *dt_tbuf
int  dt_tlen
```

The commands using this format are:

LDSETDT Sets the template buffer to contain the first **dt_tlen** characters of **dt_tbuf**, if the enhanced edit line discipline has been entered (if **c_line** equals 1, for example). At most, **DTBISIZE** characters are used. If **dt_tlen** is -1, the template buffer is not initialized.

LDGETDT Gets the current contents of the template buffer. The characters in the buffer are written starting at **dt_tbuf**, and **dt_tlen** is set to the number of characters written. At most, DTBSIZE characters will be returned. The characters will not be null-terminated.

Files

/dev/tty*
/usr/include/sys/ttmap.h

Related Information

In this book: "ioctl" on page 2-56 and "hft" on page 6-23.

trace

Purpose

Supports the event-tracing device driver.

Synopsis

#include <sys/trace.h>

Description

The **/dev/vrmtrace**, **/dev/unixtrace**, and **/dev/appltrace** files are special files that allow event records generated within the VRM, kernel, or application programs to be passed to a user program so that the activity of a driver or other system routines can be monitored for debugging purposes.

The VRM passes buffers of trace entries directly to the driver using unsolicited interrupts.

The **trace** driver supports **open, close, read**, and **ioctl** system calls. The **ioctl** system call is invoked as follows:

```
#include <sys/trace.h>
ioctl(fildes, cmd, &arg);
int fildes, cmd;

struct tr_struct
{
   unsigned channels;  /* enabled channels */
   ushort bufsize;     /* buffer size to use */
   ushort lengths;     /* communication lengths */
   char vmid;          /* VM ID of machine to trace; 0 for current */
   char timer;         /* timer to use, for VRM */
} arg;
```

Valid values of the **cmd** parameter are:

TRCSETC Sets trace parameters. This command instructs the driver to use the parameters provided in structure **arg** to set trace parameters. **bufsize** indicates the size of the buffer to allocate and cannot be changed once it is set. The **timer** field should always be a value of 0. The **channels** field is a bitmap indicating active and inactive channels. As an example, bit 0 corresponds to channel 31, bit 1 corresponds to channel 30, and bit 31 corresponds to channel 0.

TRCGETC Returns the current status of the trace in the structure indicated by **arg**.

The records returned from the trace device are structures with the following format:

```
struct
{
   unsigned stamp;                  /* time stamp */
   unsigned short timeext;          /* time stamp extension */
   unsigned short seqno[2];         /* two 16-bit sequence number digits */
   unsigned short hookid;           /* channel no. and trace event code */
   unsigned pid;                   /* process-id */
   unsigned short iodn,iocn;        /* vrm iodn/iocn or -1 */
   char data [20];                 /* more data, depending on code */
};
```

The following subchannels are assigned:

CHANNEL NUMBER	ASSIGNMENT
22	Process system calls (acct, alarm, brk, exec, fork, fstat, getgid, getgroups, getpid, getuid, kill, lockf, nice, pause, pipe, plock, profil, ptrace, reboot, setgid, setgroups, setpgrp, setuid, times, ulimit, usrinfo, utssys, wait)
23	Directory handling system calls (chdir, chroot, link, mknod, unlink)
24	I/O system calls (access, chmod, chown, close, creat, dup, fclear, fcntl, fsync, ftrunc, ioctl, lseek, open, read, umask, uname, utime, write)
25	File system system calls (mount, stat, sync, ustat, umount)

CHANNEL NUMBER	ASSIGNMENT
26	Time system calls (stime, time)
27	Signal system calls (signal, sigblock, sigcleanup, sigpause, sigsetmask, sigstack, sigvec)
28	Semaphore system calls (semctl, semget, semop)
29	Message system calls (msgctl, msgget, msgop)
30	Shared memory system calls (shmctl, shmget, shmop)
31	User-defined events

Files

/dev/vrmtrace
/dev/unixtrace
/dev/appltrace

Related Information

In this book: "trace_on" on page 3-357, "trcunix" on page 3-362, "rasconf" on page 4-133, and "Trace Logging" on page C-32.

The **trace** command in *AIX Operating System Commands Reference.*

The discussion of "trace" in *AIX Operating System Programming Tools and Interfaces.*

tty

Purpose

Supports the controlling terminal interface.

Synopsis

#include < sys/hft.h >
#include < sys/termio.h >
#include < sys/tty.h >

Description

For each process the **/dev/tty** special file is a synonym for the associated control terminal. This file is useful to programs or shell sequences that wish to ensure writing messages on the terminal regardless of how output is redirected. It can also be used for programs that demand the name of a file for output when typed output is desired, and to find out what terminal is currently in use.

Files

/dev/tty
/dev/tty*

Related Information

In this book: "hft" on page 6-23.

Chapter 7. Advanced Display Graphics Support Library

About This Chapter

This chapter describes the Advanced Display Graphics Support Library (GSL), an application programming interface to various output devices.

Subroutines, located in the **libgsl.a** library, are provided by the GSL. The **gslerrno.h** header file must be included with an **#include** statement to provide return values for the GSL subroutines.

Note: All GSL parameters are passed by reference, making the subroutines compatible with FORTRAN, in which parameters are always passed by reference. All parameters are therefore passed as pointers in C and are declared as **VAR** parameters in Pascal. The name of a GSL subroutine is always followed by an _ (underscore) in C and Pascal, but not in FORTRAN.

The **/usr/lib/samples** directory contains routines that provide clipping and transformation functions on a normalized device coordinate (NDC) system, as well as routines that provide additional function. The **README.gslext** file in this directory provides more information.

> **The Extended and Enhanced GSL subroutines that reside in the /usr/lib/samples directory are merely examples, provided for the sole purpose of illustrating that the basic GSL subroutines can be used to create extended or enhanced subroutines. The extended and enhanced GSL subroutines are each provided "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of each of the extended or enhanced GSL subroutines is with you.**

The following terms are defined for this chapter:

Frame buffer A display adapter frame buffer is memory storage containing a representation of a display image.

Geometric text The two types of text supported by the GSL are annotated, or standard, text and geometric text, which is also referred to as a programmable character set (PCS) or stroke text. For more information on annotated text, see "fonts" on page 4-68.

KSR Mode KSR (keyboard send-receive) Mode causes a virtual terminal to act like a standard ASCII terminal, with some RT PC extensions, for both input and output. See "hft" on page 6-23 for more information about KSR Mode.

Monitor Mode In Monitor Mode, a virtual terminal lets an application directly access the display adapter without conflict with the standard virtual terminal output mechanism. Further information about Monitor Mode is found under "hft" on page 6-23.

Pick device Valid only for the IBM 5081 Display Adapter, a pick device is a hardware-assisted event. An area of the screen around the active cursor is specified and ***pick*** is enabled. The adapter resets a counter to zero, then counts each graphics primitive command issued. This count identifies each output function. If a graphics primitive then intersects the area defined around the cursor, the IBM 5081 Display Adapter returns the count associated with the primitive to the system. This information appears on the GSL input ring as a ***pick event***, and helps an application determine what an operator is selecting on the screen.

Pixel A pixel, or picture element, is one point in the frame buffer or on the display.

Pixel map Also known as a ***pixmap***, this is an object that defines the characteristics of a rectangle. See "gsxblt" on page 7-139 for a list of the elements defined by a pixel map.

Ring Buffer A virtual terminal in Monitor Mode can share a ring buffer with an application and place data from input devices in the buffer. The ring buffer mechanism dramatically shortens the input data path from the virtual terminal to the application.

Overview

The GSL allows applications to perform graphics operations without the need to directly manipulate the underlying hardware. The GSL also supports the display of fixed spaced characters in text.

The GSL assumes that an application using it runs in its own virtual terminal. A virtual terminal can operate in either KSR mode (the default) or in Monitor Mode. An application may use Monitor Mode and the ring buffer to derive its own graphics interface. The GSL provides an interface that lets a user generate graphics interactively without detailed knowledge of the display adapter and input data formats. The GSL works only with the application virtual terminal in Monitor Mode. Part of the GSL initialization is to place the virtual terminal in Monitor Mode. This forces some restrictions on the use of the display adapter. The application virtual terminal can be one of several virtual terminals opened by a user, but only one virtual terminal can be ***active*** for input at any time. Several virtual terminals can be active for output at any time if multiple displays are attached, with one virtual terminal active for output on each display. All virtual terminals but one, however, are ***inactive*** for input at a given time. The active virtual terminal for input can write to the display adapter and can receive input from devices. An application must respond to user requests to become active or to release control of the display (become inactive). The transfer of control of the display occurs with two signals (a release request, **SIGRETRACT**, and a grant notification, **SIGGRANT**) and a write to the **HFT** device driver to acknowledge the release signal. After initialization, the GSL processes these two signals and writes to the device driver so that it can determine when it can and cannot write to the adapter. Routines that an application supplies that get called by the GSL signal handlers can be identified by the application during GSL initialization. The application can therefore respond appropriately to requests to be active or inactive.

The GSL provides a set of graphics output functions. Applications can supply additional functions that access the display adapter directly. Such an application routine can function only when the virtual terminal is active, and the virtual terminal must not become inactive while the routine is operating. The GSL provides a function that indicates to the application whether its virtual terminal is active or inactive, and if active, postpones GSL processing of the **SIGRETRACT** signal until the application has finished modifying the display. Another function causes the GSL to resume processing of the signal.

It is possible that one of the GSL output functions or an application-supplied output function is operating at the time of the **SIGRETRACT** signal; the function only has 30 seconds (real time) to complete the adapter operation and acknowledge the **SIGGRANT** after that signal; after the 30 seconds the **HFT** device driver sends a **SIGKILL** signal that terminates the virtual terminal. The application should be designed with this consideration in mind, or the user should be made aware of the time limit for applications that involve switching virtual terminals and have lengthy drawing operations.

The virtual terminal subsystem dictates that when a Monitor Mode virtual terminal becomes inactive and then active, the application must restore the display adapter state. At initialization the application can direct the GSL to use either of two mechanisms for restoration.

GSL Control
The GSL saves the frame buffer at the time of the **SIGRETRACT** and restores it and the appropriate adapter state, such as the color map, at the time of the **SIGGRANT**. Unfortunately, saving and restoring large frame buffers can be relatively expensive in terms of time and virtual storage space. Under this mechanism, an output operation initiated while the application virtual terminal is inactive suspends the application until its virtual terminal becomes active. If the virtual terminal is inactive when the application requests postponement of **SIGRETRACT** signal handling, the GSL suspends the application until the virtual terminal becomes active.

Application Control
The GSL saves the adapter state at the time of the **SIGRETRACT** request and calls an application routine (if provided) at the time of the **SIGGRANT**. This routine could process the applications data structure(s) to reconstruct the display adapter state. Under this mechanism, an output operation initiated while the application virtual terminal is inactive causes the output routine to return without writing to the display adapter. The routine returns a code indicating an invalid status in this circumstance.

If the virtual terminal is inactive when the application requests postponement of **SIGRETRACT** signal handling, the GSL sends a code indicating that the application cannot access the display.

Regardless of the mechanism chosen, the GSL calls an application routine (if provided) at the time of the **SIGRETRACT** request and calls an application routine (if provided) at the time of the **SIGGRANT** notification. One or both restoration routines can be chosen for an application as appropriate.

An application cannot write to standard output (using system write) on a virtual terminal that is in Monitor Mode. However, at initialization, the GSL accepts a specified file descriptor as the Monitor Mode virtual terminal from the application, and directs output to this file descriptor. An application can use more than one virtual terminal, and the virtual terminals can be mapped to different displays simultaneously. This reserves standard output for other uses such as **sdb**, the symbolic debugger.

When the ring buffer mechanism is used for processing input, the virtual terminal places input from the keyboard, locator, LPFK, valuator, or pick device in a ring buffer shared between the application and the virtual terminal. The virtual terminal causes the generation of the **SIGMSG** signal when it places the data for an input event in an empty ring buffer. At initialization, an application can select either method. However, the GSL supports only the ring buffer mechanism to optimize performance. If used, a ring buffer must be allocated by the application and made available to the GSL at initialization. The GSL sets up the virtual terminal linkage to the buffer and sets up a signal handler to catch the **SIGMSG** signal that it uses to satisfy application requests for input.

The application must then let the GSL process the ring buffer input pointer and parse the input events by invoking the appropriate input function. Whenever the application has selected the ring buffer mechanism, the application can use GSL input to enable and disable input events.

The application can provide a signal handler to catch the **SIGMSG** signal if all of the following conditions are met:

1. The signal handler is set up after the GSL is initialized.
2. The signal handler is set up using the **SIGVEC** enhanced signal function. **SIGVEC** returns the address of the GSL signal handler.
3. The signal handler must indirectly call the GSL signal handler before doing anything else. The indirect call uses the address returned by the **SIGVEC** signal.

Enhanced signals are used to block further reporting of the signal being processed until the signal handler returns. When the signal handler returns, the signal is automatically reset and unblocked.

When keyboard events are enabled, the virtual terminal puts *all* keystrokes in the ring buffer, including those that may normally have special meaning to the operating system (such as break). The application can let the system continue processing certain keystrokes by setting the virtual terminal break map.

For further information on Monitor Mode operation, see the discussion of the virtual terminal subsystem in the *Virtual Resource Manager Technical Reference.*

Attributes

A set of ***attributes*** that determine how a function works, or determine appearance characteristics on a display, govern all GSL operations affecting the frame buffer. Attributes are characteristics that do not change often and and therefore do not need to be parameters for the output functions. Some common attributes govern all output operations while others are unique to a particular category of output.

Common Attributes

Color display adapters may be considered to have multiple storage planes or layers forming the frame buffer, with each plane acting like the single frame buffer for a bilevel monochrome display. When writing a pixel into a multiplane frame buffer, one may write to all the planes or to a subset. The GSL ***plane mask*** attribute identifies which planes of the frame buffer GSL functions modify.

The color of a pixel on the display is ultimately determined by the color value of the pixel stored in the frame buffer. There are ***VLT-based adapters***, in which the pixel color value serves as an index into a video lookup table (VLT). The entry in the VLT for an index contains a value for each of the red, green, and blue digital-to-analog converters (DACs) on the adapter, which drive the color guns in the display tube. The actual color resulting from a particular pixel color value (VLT index) depends on the values loaded into the VLT, which may be any values. There are also ***true color adapters*** in which the pixel color value actually drives the DACs, without the level of indirection forced by the VLT.

An application can determine the mapping from the color used in operations on the frame buffer to the actual color shown on a display by using the GSL ***color map*** attribute. For VLT-based adapters, the GSL actually loads the adapter VLT, using color values provided by the application; the "color" used by the application is really an index into the VLT. For true color adapters, the color map serves strictly as an internal mapping from the color value specified to the actual color value loaded into the frame buffer, and the "color" used by the application is an index into the mapping table.

The application may set the color map by providing an array of color specifications; the maximum number of specifications is display adapter dependent and is determined by the number of VLT entries, or by the number of bit planes for true color adapters. The color specification for each color index comprises three intensity values, one each for the red, green, and blue DACs. Each intensity value must range from 0 - 0x3FFF. For a VLT-based display adapter, the GSL maps the color specification to the nearest available color produced by the adapter; the GSL truncates the intensity value for a color to produce a value equal in resolution to the DAC for that color.

The ***logical operation*** attribute determines how the GSL combines the pixels it generates with the current contents of the frame buffer. Sixteen Boolean combinations exist between a source (the GSL-produced pixels) and a destination frame buffer, but can only be used with the IBM 5081 Display Adapter. The GSL does, however, assure support for the most recognizable and useful Boolean combinations (replace and exclusive-or) regardless of hardware support.

The following table shows the categories of functions to which the common attributes apply.

Area	**Output**	**Pixel Block**	**Cursor**
Plane Mask	yes	yes	yes
Color Map	yes	yes	yes
Logical Operation	yes	no	no

Unique Attributes

Some unique attributes as well as the plane mask, color map, and logical operation attributes, govern the GSL functions that affect the frame buffer.

Lines

The GSL draws all lines a single pixel thick. These unique attributes that govern line drawing can be changed:

Line style Determines the pattern appearance of the line. The line style attribute provides for solid, dashed, dotted, dashed-dotted, and dashed-dotted-dotted lines, and for line patterns defined by the application.

Line color Is an index into the color map table (or VLT).

Markers

The marker attributes determine characteristics of symbols used to mark points. The GSL provides a set of predefined markers for the application to select. A marker can be custom defined by the application.

The application may change the following unique attributes that govern marker operations:

Marker color Sets the color of a marker, and is an index into the color map table (or VLT)

Marker origin Sets the point in the marker pattern that is placed at the position indicated by the application for the polymarker subroutine

Marker style Selects predefined or custom markers

Marker width Defines the width of the pattern for a custom marker

Marker height Defines the height of the pattern for a custom marker

Marker pattern Sets the form of the custom marker, and is a bit array defined by the application.

Text

The GSL places characters with a transparent background. That is, only the "strokes" in a character change data in the frame buffer. These unique attributes govern text operations and can be set by an application:

Text font Sets which of the available fonts is used for the characters

Text color Sets color and brightness of the text and is an index into the color map table or VLT

Code page Sets the page from which graphic symbols are drawn

Baseline direction

Sets the direction in which characters are written to the baseline for the text. The baseline for the string is placed at the location given in the command to write text.

Filled Areas

The edges of an area are treated as part of the area and only define the area to be filled. The GSL does not treat the edges of an area as lines. The application may change the following unique attributes that govern fill operations:

Fill color Is an index into the color map table (or VLT)

Fill pattern Is the identifier for the pattern used to fill the area.

Cursor

The GSL provides a single cursor for the application. The application may change the following unique attributes that govern cursor operations.

Cursor pattern Sets the cursor shape and is a bit array defined by the application. The minimum and maximum sizes for the cursor pattern are device-dependent and available to the application.

Cursor color Sets the cursor color and is an index into the color map table or VLT.

Cursor origin Sets the point in the cursor pattern that is placed at the position indicated by the application during cursor movement.

At GSL initialization, some of the attributes receive default values. The attributes and their default values are listed in the following table:

Attribute	Default value
color map	device dependent
plane mask	all planes enabled
logical operation	3 (replace)
line style	solid
line color (index)	7 (white)
font	device dependent
code page	P0
baseline direction	0 (left to right)
text color (index)	7 (white)
fill color (index)	7 (white)
fill pattern	0 (solid)
cursor pattern	undefined
cursor color	undefined
cursor origin	undefined

Figure 7-1. Default Attribute Values

Cursor Operations

The GSL provides one cursor for applications. The GSL cursor is non-destructive; the contents of the display adapter frame buffer remain intact when the cursor is already visible and subsequently moved or made invisible. This is achieved in a device-dependent manner. The GSL uses any hardware cursor support available.

The application totally controls the placement and visibility of the cursor. The GSL input functions do not provide cursor movement, and the GSL output routines do not check whether they are drawing over the cursor and do not automatically erase and restore the cursor or check for interference. It is therefore possible for the output routines to overwrite the cursor. When the cursor is moved on a display without hardware cursor support, any primitives that overwrite the cursor will themselves be overwritten when the area at the previous cursor position is restored. The application should erase and redraw the cursor as appropriate to avoid conflict.

Coordinate Clipping and Transformation

For simplicity and optimized performance, the base GSL does not perform general clipping or transformation on coordinates. Most of the output functions accept coordinates in the first quadrant (0,0 is the lower left corner) and convert them as necessary to the target quadrant required for the frame buffer of the specific device.

The coordinate system is device dependent, and any point outside the frame buffer range can result in a write to any address on the I/O bus. For this reason, the GSL checks coordinates sent as parameters and also coordinates generated internally against the frame buffer boundaries. *Any coordinate outside the frame buffer is invalid and produces an invalid status return.* The display results depend on the function invoked. Invalid coordinates are handled as follows:

Lines The GSL checks the input coordinates as it draws the lines. Thus, for polylines and multilines, the part of the sequence up to the invalid coordinate results in lines on the display. The functions return an error code upon encountering an invalid coordinate, drawing no further lines.

Text If the start point of the text string is invalid, the GSL returns immediately with an error code. Invalid internal coordinates may be generated if the start point is valid but part of the text string overflows the frame buffer. If the application places the baseline such that the characters must be clipped vertically (for example, the top half of the string is out of the frame buffer), the GSL writes none of the characters in the string. If the application places the baseline such that a character in the string overflows the frame buffer, that entire character and the rest of the string is truncated.

Filled Areas The GSL checks the coordinates of filled areas before it writes to the frame buffer. It returns an invalid return code for any invalid coordinate found before writing to the frame buffer.

Cursor The application may place the cursor origin anywhere within the cursor pattern. If the cursor origin is placed so that any part of the cursor falls outside of the frame buffer, an error code is returned and the cursor is not moved.

Pixel Block Transfer If any portion of the source or destination rectangle lies outside its pixmap, the GSL returns immediately with an error code.

Displays

You can use GSL support for the following all-points-addressable display adapters and displays:

6153 Display
A monochrome 720 x 512 adapter; a 12-inch display

6154 Display
A 16 of 64 color, 720 x 512 adapter; a 14-inch display

6155 Display
A monochrome 1024 x 768 adapter with significant graphics assist; a 15-inch display

5081 Display
A 256 of 4096 color, 1024 x 1024 adapter; a 16-inch or 19-inch display.

The GSL automatically uses the correct configuration for an installed display adapter at initialization. It accepts input from any device that conforms to the virtual terminal interface as described in the *Virtual Resource Manager Technical Reference.* The GSL supports one or more of the following input devices in an application:

- Keyboard
- Mouse or tablet
- Lighted Program Function Keyboard (LPFK)
- Valuator
- Pick device (valid for IBM 5081 Display Adapter only).

At least one input device is always available; the virtual terminal subsystem determines that, at minimum, keyboard input is accepted.

Printers and Plotters

Note: The printer text data stream supports only the ASCII character set.

Before an application can use GSL subroutines to generate graphic output to a printer or plotter, the Graphics Development Toolkit device drivers must be installed on the system. See the section about installing additional operating system programs in *Installing and Customizing the AIX Operating System* for instructions on how to do this.

Certain information about the device must be defined using AIX environment variables. If you enter the definitions at the shell command line, then they remain in effect only for the current login session. If you want these definitions to remain in effect for future login sessions, add them to the **.profile** file in your home directory. To define this information permanently for all users, add it to the **/etc/profile** file. See the **sh** command in *AIX Operating System Commands Reference* for more information about AIX environment variables, which are also called ***shell variables.***

1. Define the path to the Graphics Development Toolkit device drivers:

   VDIPATH = /usr/lpp/vdi/drivers
   export VDIPATH

2. Define a logical identifier for the device as an environment variable, and set its value to indicate the type of printer or plotter device:

   devname = **vdi***xxxx*
   export *devname*

   The name you use in place of *devname* can be any sequence of up to eleven alphanumeric characters. This is the name that you specify in the *fildes* parameter of the **gsinit** subroutine.

   The value of the environment variable, **vdi***xxxx*, is one of the following names:

vdi3812	IBM 3812 Printer
vdi4201	IBM 4201 Printer
vdi5152	IBM 5152 Printer
vdi5182	IBM 5182 Printer
vdi6180	IBM 6180 Plotter
vdi7371	IBM 7371 Plotter
vdi7372	IBM 7372 Plotter
vdi7375	IBM 7374, 7374-1, or 7375-2 Plotter

3. You can specify a printer or plotter as **vdi***xxxx* in step 2; **vdi***xxxx* must be associated with an AIX special file:

   vdi*xxxx* = **/dev/***yyyy*
   export vdi*xxxx*

   If you do not need output from a specific printer device, you can pipe the output to a printer queue:

   vdi*xxxx* = "| **print -plot** [*queue*]"
   export vdi*xxxx*

   Note: You can only pipe output to a queue for printer devices, not for plotters.

4. If you are using an IBM 3812 Pageprinter, then you should set and **export** (as in the examples below) the following additional environment variables:

   MARGIN Set to either **TRUE** or **FALSE**, indicating whether to leave 1/4-inch margins. If not defined, the default is **MARGIN = FALSE.**

   ORIENTATION Set to either **LANDSCAPE** or **PORTRAIT**, indicating horizontal or vertical page orientation, respectively. ***Landscape orientation*** rotates the image 90 degrees so that the horizontal axis of the image goes down the length of the page. If not defined, the default is **ORIENTATION = PORTRAIT.**

Examples

- The following example defines GRAPHDEV as the logical name of an IBM 3812 printer that is configured as **/dev/tty1**. This configuration specifies ***portrait orientation*** (output frame vertical dimensions are greater than horizontal) and no margins.

  ```
  VDIPATH=/usr/lpp/vdi/drivers
  GRAPHDEV=vdi3812
  vdi3812=/dev/tty1
  MARGIN=FALSE
  ORIENTATION=PORTRAIT
  export VDIPATH GRAPHDEV vdi3812 MARGIN ORIENTATION
  ```

- The next example defines GRAPHDEV as the logical name of an IBM 3812 printer that is already configured as the device that serves printer queue **lp0**. This configuration specifies ***landscape orientation*** (output frame horizontal dimensions are greater than vertical) and 1/4-inch margins.

  ```
  VDIPATH=/usr/lpp/vdi/drivers
  GRAPHDEV=vdi3812
  vdi3812="| print -plot lp0"
  MARGIN=TRUE
  ORIENTATION=LANDSCAPE
  export VDIPATH GRAPHDEV vdi3812 MARGIN ORIENTATION
  ```

Note: Only the output subroutines, listed below, are valid for printers and plotters.

Functional Categories of Subroutines

The base GSL is device dependent in that it provides some functions for the IBM 5081 Display that it does not provide for other displays. It also does not scale, clip, or transform coordinates for any display. Coordinates passed to the base GSL functions are therefore device dependent, and limited to the device boundaries. The GSL does make device specific information available through query commands, letting an application perform appropriate clipping and transformation. It also indicates the logical operations supported by the hardware.

The base GSL is organized into several major function areas:

- Control
- Output
- Service
- Pixel Block Transfer
- Cursor

- Attribute
- Input
- Query.

The following sections provide an overview of the functions in each area.

Control

The base GSL provides functions to initialize and terminate the GSL and to coordinate direct application access to the display device. At initialization, the GSL sets up its required environment and establishes ***Monitor Mode*** on the application virtual terminal. Monitor Mode provides the GSL with direct access to the display adapter without interference from the virtual terminal subsystem. Monitor Mode operation also gives faster access to input event information. At termination, the GSL cleans up after itself and returns the application virtual terminal to KSR mode.

These subroutines perform overall control operations of the GSL environment.

gsinit	Initializes the GSL subroutines, establishes Monitor Mode on the application virtual terminal, and allows specification of application-supplied signal processing routines.
gslock	Locks the virtual terminal so that the application can access the display adapter directly.
gsterm	Terminates the GSL, returns the application virtual terminal to KSR mode.
gsunlk	Unlocks the virtual terminal, returning control to the GSL.

Output

The GSL output functions provide an application with capabilities to perform graphics operations on output devices. The output functions can be divided into these categories:

Drawing lines

The GSL provides functions to draw:

A line between two points
A series of lines connecting a sequence of points
A series of lines connecting alternate pairs in a sequence of points.

GSL lines are a single pixel thick. Specific attributes allow lines of different colors and patterns.

Drawing polymarkers

The polymarker subroutine in the GSL lets a defined marker be drawn for a sequence of points. The definition of the marker includes specific attributes such as color, style, width, height, pattern, and origin. The pattern attribute is a raster image to be used as a marker. The origin attribute controls the placement of the polymarker pattern at the points specified by the polymarker subroutine.

Writing annotated text
The GSL provides a function to write a text string to the display adapter at a given starting position. Character placement is with a transparent background so that the GSL changes only the character shape (foreground), not the entire character box. Specific attributes allow text in different fonts, colors, code pages, and directions.

Writing geometric text
The GSL provides functions to write geometric text strings. This capability is provided only for use with the IBM 5081 Display, and not for use with other displays.

Drawing curves
The GSL provides functions to draw circles, arcs, and ellipses. These functions are used to achieve the best performance and quality possible so that your programs can realize the full capability of your display.

Filling areas
The GSL provides functions to draw filled rectangles and general polygons, circles, and ellipses. In addition, the GSL allows curves to be combined with polylines to fill complex shapes. See “gsbply” on page 7-20, “gseply” on page 7-50, and “gspcls” on page 7-106. These functions allow applications to use the higher performance possible with rectangles. The GSL also provides a **color zero** function to clear the display to the background color. Specific attributes allow different colors and patterns. See ″Attributes″, on page 7-6.

Each category of the output functions is governed by a set of attributes. Some attributes determine characteristics that are specific to the category, such as color or pattern. These attributes are common to all categories:

color table
Maps color names or values to the actual color on the display (see “Common Attributes” on page 7-6)

plane mask
Determines which of the display adapter storage planes are modified by the output functions

logical operation
Determines how the GSL combines the foreground or background color for each pixel produced by a primitive with the current color of the destination pixel in the frame refresh buffer. This attribute is not valid for the IBM 5081 Display Adapter.

These output subroutines write to the display adapter frame buffer, generally producing output on a display screen:

gsbply	Begins a polygon.
gscarc	Draws a circular arc of a specified radius between two points.
gscir	Draws a circle.

gsclrs Clears the display screen, filling it with the background color.

gscrca Draws a circular arc between two angles.

gseara Draws an elliptical arc between two angles.

gsearc Draws an elliptical arc of specified axes and angle between two points.

gsell Draws an ellipse.

gseply Ends a polygon.

gsfci Fills a circle.

gsfell Fills an ellipse.

gsfrec Draws a filled rectangle.

gsfply Draws a filled polygon.

gsgtxt Displays a geometric text string, with NDC transformations supported.

gsline Draws a line between two points.

gsmult Draws a multiline, or a set of straight lines that connect alternate pairs of points in a sequence.

gspcls Closes a polygon.

gsplym Draws a polymarker, a marker (such as a dot or plus sign) at each of a specified set of points.

gspoly Draws a polyline, or a path of straight lines that connect a sequence of points.

gstext Displays a text string.

Service

The GSL provides functions for defining a circular or elliptical arc. These functions convert circular or elliptical arc definitions into sets of vertices. The resulting set of vertices can be drawn, using the **gsline** subroutine, or combined with other polylines to draw or fill more complex shapes.

The attributes that can be used for drawing lines or filling areas apply here, including style, color, logical operation, pattern, and others.

Arcs are specified by beginning and ending points or beginning and ending angles and follow the counterclockwise direction. If the beginning and ending points are identical, then the list of vertices corresponding to a full circle or ellipse is returned. This allows circles or ellipses to be treated as a special case of closed arcs. If off-axis, ellipsis angle is specified in degrees. There are four levels of precision for the conversion of an arc into a set of line segments.

These subroutines facilitate the drawing of circular and elliptical arcs.

gsccnv Converts a circle to a set of vertices (polyline).

gsecnv Converts an ellipse to a set of vertices (polyline).

Pixel Block Transfer

The GSL provides functions to move a rectangular block of pixels from either the display adapter frame buffer or storage to either the display adapter or storage. If the source rectangle or destination rectangle reside in a color display adapter frame buffer, this operation is affected by the plane mask attribute. If the destination rectangle resides in a color display adapter frame buffer, this operation is affected by the color map attribute.

These subroutines allow a program to:

- save a block of pixels from the frame buffer
- restore a block of pixels from the frame buffer
- move a rectangular shape from adapter memory to pixel memory
- move a rectangular shape from adapter memory to system memory
- move a rectangular shape from one area in system memory to another
- move a rectangular shape from one area of adapter memory to another
- move a tile rectangle to any area of visible pixel memory.

gsrrst	Restores a rectangular block.
gsrsav	Saves a rectangular block.
gsxblt	Moves a rectangular block from one location in memory or display adapter frame buffer to another location in memory or display adapter frame buffer.
gsxcnv	Converts pixel format data to and from plane format data.
gsxptr	Handles FORTRAN addressing of pixel map data.

Cursor

The GSL provides functions to draw and undraw a non-destructive cursor. The application is responsible for the placement and visibility of the cursor. The input functions do not provide for cursor movement, nor do output or pixel block transfer functions check whether they interfere with the cursor. Anything unintentionally placed over the cursor is modified when the cursor moves. The color map and plane mask attributes govern the cursor functions. Cursor pattern and color can be defined by attributes.

These are the cursor subroutines:

gsecur	Erases the cursor and makes it invisible.
gsmcur	Moves the cursor and makes it visible.

Attribute

The GSL provides functions to set the global attributes and all of the output category specific attributes. The GSL also provides functions to set attributes of some of the input devices.

These subroutines set attributes for both input and output operations:

gscatt	Sets the cursor attributes.

gscmap	Sets the color map.
gsfatt	Sets the fill attributes.
gsgtat	Sets the attributes for the geometric text drawing operation, **gsgtxt**.
gslatt	Sets the line attributes.
gslcat	Sets the locator attributes.
gslpat	Sets the LPFK indicators.
gslop	Sets the logical operation used for drawing lines.
gsmask	Sets the plane mask.
gsmatt	Sets the attributes for the polymarker operation, **gsplym**.
gspp	Sets plotter pen speed as a percentage of the plotter maximum speed.
gstatt	Sets the attributes for the text output operation, **gstext**.
gsulns	Sets the user line pattern.
gsvgrn	Sets the valuator granularity.

Input

An application using the GSL can receive input with the standard **read** system call or through a faster mechanism available through the virtual terminal. While the GSL allows an application to use the standard mechanism, it provides no input support for it.

The GSL accepts input from several sources:

- The keyboard
- The locator, which can be a mouse or a tablet
- The lighted programmed function keys (LPFKs)
- The valuator dials
- Pick device (valid for IBM 5081 Display Adapter only).

Input from these devices is viewed as discrete events, with input data associated with each event.

The GSL provides subroutines to enable or disable input from any device, and a subroutine that lets a program suspend execution until one of the enabled events occurs. The latter subroutine also parses the raw data generated by the virtual terminal and makes the parsed information available to the application. In addition, two GSL subroutines allow you to enable or disable pick events.

The state established (enabled or disabled) remains in effect when the GSL terminates. Note that for these subroutines to work properly, a valid input ring buffer must have been specified to the **gsinit** subroutine.

gsdpik	Disables picking.
gsepik	Enables picking.
gsevds	Disables the reporting of input events.
gseven	Enables the reporting of input events from the keyboard, locator, LPFK, or valuator.
gsevwt	Waits for an input event and parses the raw data.

Query

The GSL provides functions for applications to query the active display adapter characteristics, the currently active annotated or geometric text font, and some input device characteristics. Through query functions an application can derive the information necessary to deal with any device dependencies. Note that **gsinit** must be invoked before calling any of the query subroutines.

These are subroutines that provide query functions:

gsqdsp	Returns device-specific information about the display adapter and display monitor.
gsqfnt	Returns information about the current annotated text font.
gsqgtx	Returns information about the current geometric text font.
gsqloc	Returns device-specific information about the locator.

gsbply

Purpose

Defines the beginning of an area to fill.

C Syntax

int gsbply_ ( )

FORTRAN Syntax

INTEGER function gsbply

Pascal Syntax

FUNCTION gsbply_ : INTEGER [PUBLIC];

Description

The **gsbply** subroutine defines the beginning of a two-dimensional shape or set of shapes to be filled.

The following output routines are valid between a **gsbply** call and a **gseply** call:

• Draw polyline	(**gspoly**)
• Draw circle	(**gscir**)
• Draw ellipse	(**gsell**)
• Draw circular arc	(**gscarc** or **gscrca**)
• Draw elliptical arc	(**gseara** or **gsearc**)

Note: Any other subroutines used before the **gseply** subroutine is called do not become part of the shape or set of shapes to be filled, and can produce unpredictable results.

Before the fill occurs, the shapes drawn by each routine called between **gsbply** and **gseply** are connected. The first point of each shape is linked to the last point of the previous shape, and the last point of the last shape is linked to the first point of the first shape. The shapes may overlap to any degree but must share at least one common point between adjacent shapes.

Processing of the **SIGRETRACT** signal is postponed until the **gseply** subroutine, end of area to fill, is called.

See "gseply" on page 7-50 and "gspcls" on page 7-106 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: "gseply" on page 7-50 and "gspcls" on page 7-106.

gscarc

Purpose

Draws a circular arc between two points.

C Syntax

int gscarc_ (*cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey***)**

int **cx*, **cy*, **cr*, **bx*, **by*, **ex*, **ey*;

FORTRAN Syntax

INTEGER function gscarc (*cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey***)**

INTEGER *cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*

Pascal Syntax

FUNCTION gscarc_ (

VAR *cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gscarc** subroutine draws a counterclockwise circular arc of the specified radius from the beginning point to the ending point. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of the circle. For displays, the center is restricted to -2048 to 2048. For printers and plotters, the center is restricted to screen coordinates.
cr	Defines the radius of the circle.
bx, by	Define the coordinates of the beginning point on the circle.
ex, ey	Define the coordinates of the ending point on the circle.

If the beginning and ending points are identical, a full circle is drawn.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_RDUS	Invalid radius specification.
GS_INAC	Virtual terminal inactive.
GS_AEND	Invalid end point.
GS_ASTR	Invalid start point.

gscatt

Purpose

Sets the cursor attributes.

C Syntax

int gscatt_ (*color*, *width*, *height*, *pattern*, *0x*, *0y***)**

int **color*, **width*, **height*, **pattern*, **0x*, **0y*;

FORTRAN Syntax

INTEGER function gscatt (*color*, *width*, *height*, *pattern*, *0x*, *0y***)**

INTEGER *color*, *width*, *height*, *pattern*, *0x*, *0y*

Pascal Syntax

FUNCTION gscatt_ (

VAR *color*, *width*, *height*: **INTEGER;**
pattern: **ARRAY [1..*k*] of INTEGER;**
0x, *0y*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gscatt** subroutine defines the cursor for the GSL. The **gscmap** subroutine must initialize the color map before **gscatt** can be called.

Parameters

color	Refers to an entry in the color map. If the index value is -1, the attribute is unchanged.
width, height	Define, in pixels, the width and height of the bit pattern to be used as the cursor. If *width* or *height* equals -1, then the pattern remains unchanged.

pattern	Defines the image used as a cursor. The ceiling (*width*/32) indicates the number of words per row and *height* indicates the number of rows. The cursor data must be supplied in row (scan line) major order. If *width* implies partial use of a word, the rest of the word is unused. To fully define the cursor pattern, *pattern* should be (ceiling(*width*/32)×*height*) words in length.
0x, *0y*	Indicate the origin of the cursor relative to the lower leftmost corner (0, 0) of the cursor pattern. The origin must be placed within the cursor pattern: $0x < width$ and $0y < height$. The origin of the cursor is placed at the position indicated, when the application moves the cursor using the **gsmcur** subroutine. If *x* equals -1, then the origin remains unchanged.

The maximum size of the cursor is device dependent and can be determined by using the **gsqdsp** subroutine.

You cannot change the cursor attributes while the cursor is visible.

There is no default cursor defined, so all cursor parameters must be set before the cursor is displayed.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_CURS	Cursor size invalid.
GS_CURO	Cursor origin invalid.
GS_CURV	Cursor visible.

gsccnv

Purpose

Converts a circular arc or full circle into a polyline.

C Syntax

int gsccnv_ (*cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *len*, *x*, *y*, *pre***)**

int **cx*, **cy*, **cr*, **bx*, **by*, **ex*, **ey*, **len*, **x*, **y*, **pre*;

FORTRAN Syntax

INTEGER function gsccnv (*cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *len*, *x*, *y*, *pre***)**

INTEGER *cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *len*, *x***(*)**, *y***(*)**, *pre*

Pascal Syntax

FUNCTION gsccnv_ (

VAR *cx*, *cy*, *cr*, *bx*, *by*, *ex*, *ey*, *len*: **INTEGER;**
VAR *x*, *y*: **ARRAY** [1..*k*] **of INTEGER;**
VAR *pre*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsccnv** subroutine converts a counterclockwise circular arc definition into an array of vertices. The list of vertices can then be used to draw a circular arc with the **gspoly** subroutine or to fill a circular arc with the **gsfply** subroutine. In general, it can be concatenated with other list(s) of vertices to draw or fill more complex shapes, such as chord arcs, pie arcs, and rectangles with rounded corners.

When beginning and ending points are identical, the list of vertices contains the full circle, which can then be drawn or filled.

Parameters

cx, cy Define the coordinates of the center of the circle.

cr Defines the radius of the circle, which must not be equal to zero.

If *cr* is negative, it is automatically converted to a positive value for use by the subroutine.

bx, by Define the coordinates of the beginning point of the arc.

ex, ey Define the coordinates of the ending point of the arc.

len Defines the number of points in the coordinate *x* and *y* arrays. It must be numerically at least one greater than the value contained in the precision parameter, but not less than 65.

x, y Define, as coordinate arrays, the vertices that represent the circular shape when drawn or filled.

pre Defines precision level, which specifies the maximum number of line segments that can be generated for a full circle. The number of line segments actually generated depends on the size of the circle.

There are four levels of precision that can be requested:

- 64 (65 vertices)
- 128 (129 vertices)
- 256 (257 vertices)
- 512 (513 vertices).

Therefore, *len* ≥ *pre* + 1.

All other precision values are reserved and must not be used, as their results are unpredictable. The default value for *pre* is 64.

The subroutine allows ample leniency toward the accuracy of the specification of the beginning and ending points. The arc of the specified radius will always start and end exactly at the specified points.

If the beginning and ending points are identical, a full circle of the specified radius is generated.

When the subroutine is invoked, the length parameter must contain the maximum number of entries in the *x* and *y* arrays. If erroneous conditions arise, *len* is set to zero. Under normal conditions, *len* specifies the number of vertices returned by the subroutine in the *x* and *y* arrays.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.

gscir

Purpose

Draws a circle.

C Syntax

int gscir_ (*cx*, *cy*, *cr***)**

int **cx*, **cy*, **cr*;

FORTRAN Syntax

INTEGER function gscir (*cx*, *cy*, *cr***)**

INTEGER *cx*, *cy*, *cr*

Pascal Syntax

FUNCTION gscir_ (

VAR *cx*, *cy*, *cr*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gscir** subroutine draws a circle of the specified radius. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of the circle.
cr	Defines the radius of the circle. If the radius is zero, a single point is drawn at the center.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_RDUS	Invalid radius specification.
GS_INAC	Virtual terminal inactive.

gsclrs

Purpose

Clears the screen, filling it with the background color.

C Syntax

int gsclrs_ ( )

FORTRAN Syntax

INTEGER function gsclrs

Pascal Syntax

FUNCTION gsclrs_: INTEGER [PUBLIC];

Description

The **gsclrs** subroutine fills the frame buffer with the background color (color index zero).

The relevant attribute is:

- Color map.

For printers, the **gsclrs** subroutine forces pending graphics to be printed, advances the paper to a new page, and purges the print buffer.

For plotters, the **gsclrs** subroutine forces pending graphics to be *displayed*, and issues a prompt to the active screen (console) requesting that the paper be changed.

Return Value

GS_SUCC	Successful.
GS_INAC	Virtual terminal inactive.

gscmap

Purpose

Specifies the color mapping.

C Syntax

int gscmap_ (*number*, *red*, *green*, *blue***)**

**int ****number*, ***red*, ***green*, ***blue*;

FORTRAN Syntax

INTEGER function gscmap (*number*, *red*, *green*, *blue***)**

INTEGER *number*, *red* **(*)**, *green* **(*)**, *blue* **(*)**

Pascal Syntax

FUNCTION gscmap_ (

VAR *number* **INTEGER;**
VAR *red*, *green*, *blue*: **ARRAY [0..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gscmap** subroutine specifies the mapping between the color index attribute and the color it produces on the display.

The default color table mapping for the first 16 colors is the same as the default color map attributes in KSR mode. The remaining color values are initialized in a hardware dependent manner.

Parameters

number Indicates how many colors the input intensity arrays contain.

red, green, blue Define arrays that contain the intensity levels of the corresponding color. Each entry in an array specifies the intensity value for the corresponding color index.

The value in each entry for the *red*, *green*, and *blue* intensity arrays is between 0x0000, representing zero intensity, and 0x3FFF, representing full intensity. The following additional increments of intensity are possible, depending on the adapter hardware in use:

0x2000	1/2 intensity
0x1000	1/4 intensity
0x0800	1/8 intensity
0x0400	1/16 intensity
0x0200	1/32 intensity
0x0100	1/64 intensity.

Combinations of these values can be used to create intermediate levels of intensity. For example, 0xC000 gives 3/16 intensity, while 0x3000 gives 3/4 intensity.

The actual number of bits from bit 13 to bit 0 that affect the color on the display is dependent on the number of bits in the digital-to-analog converter of the adapter hardware in use. This size information is available by using the **gsqdsp** subroutine.

An application cannot change a single arbitrary color entry in the color map (or the VLT). It must change all the entries for all the colors up to and including the desired entry.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_TABL	Invalid table length.
GS_INAC	Virtual terminal inactive.

gscrca

Purpose

Draws a circular arc between two angles.

C Syntax

int gscrca_ (*cx*, *cy*, *cr*, *ba*, *ea***)**

int **cx*, **cy*, **cr*, **ba*, **ea*;

FORTRAN Syntax

INTEGER function gscrca (*cx*, *cy*, *cr*, *ba*, *ea***)**

INTEGER *cx*, *cy*, *cr*, *ba*, *ea*

Pascal Syntax

FUNCTION gscrca_ (

VAR *cx*, *cy*, *cr*, *ba*, *ea* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gscrca** subroutine draws a counterclockwise circular arc of the specified radius from the beginning point as defined by an angle specification to the ending point as defined by an angle specification.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gscrca** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of the circle. For displays, the center is restricted to -2048 to 2048. For printers and plotters, the center is restricted to screen coordinates.
cr	Defines the radius of the circle in device coordinates.
ba	Defines the start point of the circular arc as an angle in tenths of degrees, from 0 to 3600.
ea	Defines the end point of the circular arc as an angle in tenths of degrees, from 0 to 3600.

If the beginning and ending angles are identical, a full circle is drawn.

Return Value

GS_SUCC	Successful.
GS_ANGL	Invalid angle.
GS_RDUS	Invalid radius specification.
GS_CORD	Invalid coordinate.
GS_INAC	Virtual terminal inactive.

gsdpik

Purpose

Defines the closing delimiter for a group of GSL output functions.

C Syntax

int gsdpik_ ( )

FORTRAN Syntax

INTEGER function gsdpik

Pascal Syntax

FUNCTION gsdpik_ : INTEGER [PUBLIC];

Description

The **gsdpik** subroutine defines the closing delimiter for a group of pickable output functions. The output function calls that precede this command cause a pick input from the display adapter if any vertices intersect a pick aperture (window).

The **gsdpik** subroutine is provided only for use with the IBM 5081 Display, and not for use with other displays.

See "gsepik" on page 7-48 and the list of GSL output subroutines on page 7-15 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: "gsepik" on page 7-48 and the list of GSL output subroutines on page 7-15.

gseara

Purpose

Draws an elliptical arc between two angles.

C Syntax

int gseara_ (*cx*, *cy*, *ma*, *mi*, *ang*, *sa*, *ea***)**

int **cx*, **cy*, **ma*, **mi*, **ang*, **sa*, **ea*;

FORTRAN Syntax

INTEGER function gseara (*cx*, *cy*, *ma*, *mi*, *ang*, *sa*, *ea***)**

INTEGER *cx*, *cy*, *ma*, *mi*, *ang*, *sa*, *ea*

Pascal Syntax

FUNCTION gseara_ (

VAR *cx*, *cy*, *ma*, *mi*, *ang*, *sa*, *ea* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gseara** subroutine draws a counterclockwise elliptical arc of the specified axes and angle from the beginning point defined by an angle specification to the ending point defined by an angle specification. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gseara** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy Define the coordinates of the center of the ellipse.

For displays, the center is restricted to -2048 to 2048.

For printers and plotters, the center is restricted to screen coordinates.

ma, mi Define half of the non-zero major and minor axes of the ellipse.

ang Defines the angle between the major axis and the x-axis. If *ang* is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is expressed in tenths of degrees, from 0 to 3600.

sa Defines the angle of the starting point of the elliptical arc, measured counterclockwise from the major axis. The angle is expressed in tenths of degrees, from 0 to 3600.

ea Defines the angle of the ending point of the elliptical arc, measured counterclockwise from the major axis. The angle is expressed in tenths of degrees, from 0 to 3600.

If the beginning and ending points are identical, a full ellipse is drawn.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_ELMM	Invalid major or minor axis.
GS_INAC	Virtual terminal inactive.
GS_ANGL	Invalid angle.
GS_NMEM	Insufficient resources.

gsearc

Purpose

Draws an elliptical arc between two points.

C Syntax

int gsearc_ (*cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot***)**

int **cx*, **cy*, **ma*, **mi*, **ang*, **bx*, **by*, **ex*, **ey*, **rot*;

FORTRAN Syntax

INTEGER function gsearc (*cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot***)**

INTEGER *cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot*

Pascal Syntax

FUNCTION gsearc_ (

VAR *cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsearc** subroutine draws a counterclockwise elliptical arc of the specified axes and angle from the beginning point to the ending point. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsearc** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy Define the coordinates of the center of the ellipse.

For displays, the center is restricted to -2048 to 2048.

For printers and plotters, the center is restricted to screen coordinates.

ma, mi Define half of the non-zero major and minor axes of the ellipse.

ang Defines the angle between the major axis and the x-axis. If *ang* is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is expressed in tenths of degrees, from 0 to 3600.

bx, by Define the coordinates of the beginning point on the ellipse.

ex, ey Define the coordinates of the ending point on the ellipse.

rot Specifies whether the application must perform rotational transformation. Possible setting are:

0 The coordinates of the beginning and ending points passed by the application correspond to an arc of an orthogonal ellipse. No rotational transformation is performed, thus improving performance.

1 The beginning and ending points are transformed by the application and lie on the off-axis ellipse.

All other values are reserved and must not be used, as they may produce unpredictable results.

If the beginning and ending points are identical, regardless of whether or not they are on the ellipse, a full ellipse is drawn.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_ELMM	Invalid major or minor axis.
GS_INAC	Virtual terminal inactive.
GS_ANGL	Invalid angle.
GS_NMEM	Insufficient resources.
GS_AEND	Invalid end point.
GS_ASTR	Invalid start point.

gsecnv

Purpose

Converts an ellipse to a polyline.

C Syntax

int gsecnv_ (*cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot*, *len*, *x*, *y*, *pre***)**

int **cx*, **cy*, **ma*, **mi*, **ang*, **bx*, **by*, **ex*, **ey*, **rot*, **len*, **x*, **y*, **pre*;

FORTRAN Syntax

INTEGER function gsecnv (*cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot*, *len*, *x*, *y*, *pre***)**

INTEGER *cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot*, *len*, *x*(*), *y*(*), *pre*

Pascal Syntax

FUNCTION gsecnv_ (

VAR *cx*, *cy*, *ma*, *mi*, *ang*, *bx*, *by*, *ex*, *ey*, *rot*, *len*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER;**
VAR *pre*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsecnv** subroutine converts a counterclockwise elliptical arc definition into an array of vertices. The list of vertices can then be used to draw an elliptical arc with the **gspoly** subroutine or to fill an elliptical arc with the **gsfply** subroutine. In general, it can be concatenated with other list(s) of vertices to draw or fill more complex shapes, such as chord arcs, pie arcs, or rectangles with round corners.

When the beginning and ending points are identical, the list of vertices contains the full ellipse, which can then be drawn or filled.

Parameters

cx, cy Define the coordinates of the center of the ellipse.

ma, mi Define half of the non-zero major and minor axes of the ellipse.

ang Defines the off-axis angle of the ellipse. If *ang* is zero, the major axis is the x-axis and the minor axis is the y-axis. A positive value rotates the ellipse counterclockwise; a negative value rotates it clockwise. All values are in degrees and modulo 360.

bx, by Define the coordinates of the beginning point of the arc.

ex, ey Define the coordinates of the ending point of the arc.

rot Specifies whether the application must perform rotational transformation. Possible setting are:

0 The coordinates of the beginning and ending points passed by the application correspond to an arc of an orthogonal ellipse. No rotational transformation is performed, thus improving performance.

1 The beginning and ending points are transformed by the application and lie on the off-axis ellipse.

All other values are reserved and must not be used, as they may produce unpredictable results.

len Defines the number of points in the coordinate *x* and *y* arrays. It must be numerically at least one greater than the value contained in the precision parameter and greater than or equal to 65.

x, y Define, as coordinate arrays, the vertices that represent the elliptical shape when drawn or filled.

pre Defines precision level, which specifies the maximum number of line segments that can be generated for a full ellipse. The number of line segments actually generated depends on the size of the ellipse.

There are four levels of precision that can be requested:

- 64 (65 vertices)
- 128 (129 vertices)
- 256 (257 vertices)
- 512 (513 vertices).

Therefore, $len \geq pre + 1$.

All other precision values are reserved and must not be used, as their results are unpredictable. The default value for *pre* is 64.

The subroutine allows ample leniency toward the accuracy of the specification of the beginning and ending points. The arc of the specified angle always starts and ends exactly at the specified points. If the beginning and ending points are identical, a full ellipse of the specified angle is generated.

When the subroutine is invoked, the length parameter must contain the maximum number of entries in the *x* and *y* arrays. If erroneous conditions arise, *len* is set to zero. Under normal conditions, *len* specifies the number of vertices returned by the subroutine in the *x* and *y* arrays.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.

gsecur

Purpose

Erases the cursor, making it invisible.

C Syntax

int gsecur_ ( )

FORTRAN Syntax

INTEGER function gsecur

Pascal Syntax

FUNCTION gsecur_: INTEGER [PUBLIC];

Description

The **gsecur** subroutine makes the cursor invisible.

For adapters with hardware cursor support, **gsecur** simply turns off the cursor. Otherwise, **gsecur** reverses the actions taken to place the cursor in the frame buffer.

Return Value

GS_SUCC	Successful.
GS_INAC	Virtual terminal inactive.

gsell

Purpose

Draws an ellipse.

C Syntax

int gsell_ (*cx*, *cy*, *ma*, *mi*, *ang*)

int **cx*, **cy*, **ma*, **mi*, **ang*;

FORTRAN Syntax

INTEGER function gsell (*cx*, *cy*, *ma*, *mi*, *ang*)

INTEGER *cx*, *cy*, *ma*, *mi*, *ang*

Pascal Syntax

FUNCTION gsell_ (

VAR *cx*, *cy*, *ma*, *mi*, *ang* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsell** subroutine draws an ellipse of the specified axes and angle. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsell** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of the ellipse.
ma, mi	Define half of the non-zero major and minor axes of the ellipse.
ang	Defines the angle between the major axis and the x-axis. If it is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is expressed in tenths of degrees, from 0 to 3600.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_ELMM	Invalid major or minor axis.
GS_INAC	Virtual terminal inactive.
GS_ANGL	Invalid angle.
GS_NMEM	Insufficient resources.

gsepik

Purpose

Defines the beginning delimiter for a group of GSL output primitive functions.

C Syntax

int gsepik_ (*pickwind***)**

**int ****pickwind*;

FORTRAN Syntax

INTEGER function gsepik (*pickwind***)**

INTEGER *pickwind*

Pascal Syntax

FUNCTION gsepik_ (

VAR *pickwindow* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsepik** subroutine defines the opening delimiter for a group of output routines. The GSL output functions that occur after this command cause a pick input from the display adapter for each intersection of a vertex and the pick aperture window centered about the current cursor position.

The pick input is placed on the user input ring, indicating the count of output functions since the start of the pick. The input also indicates the center of the pick window in x,y coordinates.

The **gsepik** subroutine is provided only for use with the IBM 5081 Display, and not for use with other displays.

See "gsdpik" on page 7-36 and the list of GSL output subroutines on page 7-15 for related information.

Parameters

pickwind	Defines a pick window center about the current cursor position. Valid values are between 1 and the maximum cursor size.

A cursor must be defined and visible for the **gsepik** function to operate.

Processing of the **SIGRETRACT** signal is suspended until the completion of the pick function with the **gsdpik** subroutine.

Return Value

GS_SUCC	Successful.
GS_NCOR	Undefined cursor.
GS_CURV	Cursor not visible.
GS_IVWD	Invalid pick window size.
GS_USUC	Unsuccessful.

Related Information

In this book: "gsdpik" on page 7-36 and the list of GSL output subroutines on page 7-15.

gseply

Purpose

Defines the end of an area to fill.

C Syntax

int gseply_ ( )

FORTRAN Syntax

INTEGER function gseply

Pascal Syntax

FUNCTION gseply_ : INTEGER [PUBLIC];

Description

The **gseply** subroutine defines the end of a two-dimensional shape or set of shapes to be filled, then fills each of the valid primitives drawn since the last **gspcls** or **gsbply** subroutine was called.

See "gsbply" on page 7-20 and "gspcls" on page 7-106 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: "gsbply" on page 7-20 and "gspcls" on page 7-106.

gsevds

Purpose

Disables the reporting of events.

C Syntax

int gsevds_ (*event***)**

int **event*;

FORTRAN Syntax

INTEGER function gsevds (*event***)**

INTEGER *event*

Pascal Syntax

FUNCTION gsevds_ (

VAR *event*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsevds** subroutine disables the reporting of events of a given type. When the keyboard event is disabled, the keyboard is locked and no keystroke input is placed in the input ring buffer. Similarly, for all other devices, if an event is disabled, the device producing the event is inhibited from placing input into the ring.

A valid input ring must be defined during the GSL initialization.

Parameters

event The recognized events are as follows:

1 Keystroke
3 Locator movement or button
4 Lighted Program Function Key (LPFK)
5 Valuator
6 Key code.

The user can enable the keyboard by keying the sequence `ESC b` (the ANSI Enable Manual Input). After this sequence, keystroke events are again reported.

Return Value

GS_SUCC	Successful.
GS_EVNT	Invalid event type.
GS_UNSC	Unsuccessful.

gseven

Purpose

Enables the reporting of events.

C Syntax

int gseven_ (*event***)**

**int ****event*;

FORTRAN Syntax

INTEGER function gseven (*event***)**

INTEGER *event*

Pascal Syntax

FUNCTION gseven_ (

VAR *event*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gseven** subroutine enables the reporting of events of a given type. If the device producing the event is enabled, then **gseven** lets it put data into the ring buffer. If the event type is not recognized, no action is taken.

A valid input ring must be defined during the GSL initialization.

Parameters

event The recognized events are as follows:

1 Keystroke
3 Locator movement or button
4 Lighted Program Function Key (LPFK)
5 Valuator
6 Key code.

After GSL initialization, only the keyboard is enabled. If the application wishes the other input devices enabled, it must explicitly enable them with this command.

Return Value

GS_SUCC	Successful.
GS_EVNT	Invalid event type.
GS_UNSC	Unsuccessful.

gsevwt

Purpose

Waits for an input event.

C Syntax

int gsevwt_ (*wait*, *data***)**

**int ****wait*, *data*[13];

FORTRAN Syntax

INTEGER function gsevwt (*wait*, *data***)**

INTEGER *wait*, *data* **(13)**

Pascal Syntax

FUNCTION gsevwt_ (

VAR *wait*: **INTEGER;**
VAR *data*: **ARRAY [0..12] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsevwt** subroutine returns the relevant information for the oldest input event in the ring buffer.

The function works as follows:

- If an event is in the ring, then **gsevwt** parses the oldest event in the ring. It returns the event type and its data in the buffer provided by the application.
- If no event is in the ring and the application requested no wait, **gsevwt** returns immediately. If the application requested a wait, the process execution is suspended until an enabled input event occurs; then **gsevwt** returns the event type and its data in the buffer provided.

> **Warning: gsevwt** uses the application buffer passed to it for temporary storage. If the user has explicitly keyed part of an ANSI control sequence when the application calls **gsevwt** with no wait request, then **gsevwt** finds a partial event in the ring and leaves part of the parsed data for the event in the application buffer; however, **gsevwt** returns a timeout event class. Unless the application returns the same unmodified buffer, or a different buffer containing identical information, the results of the next call to **gsevwt** will be incorrect.

A valid input ring must be defined during the GSL initialization.

Parameters

wait Determines whether or not to wait for an event. If *wait* is 0, then **gsevwt** does not wait for an event if no event is available.

data Specifies the location where GSL is to store the input data (up to 13 words). The *data* must be word aligned:

The possible events are:

1 Keystroke(s)

This event type occurs when the user types a single graphic character or a single-byte control character. For these two events, **gsevwt** returns a null-terminated byte string representing the graphic or control character that was typed. This event may also occur if the user has explicitly keyed an ANSI escape sequence; **gsevwt** returns two bytes, the ESC and the next character in the sequence.

The *data* consists of a null-terminated ASCII string and is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 1			
Reserved			
code	code	code	code
		⋮	
code	code	0	unused

2 Control sequence

This event type indicates an ANSI control sequence, which is of the form:

ESC [*p* ; *p* ; . . . *p f*

where ESC is the ASCII escape character, *p* represents a parameter (one or more ASCII digits), the ellipsis represents additional parameters separated by semicolons, and *f* represents the final character that terminates the sequence (ASCII **a-z** or **A-Z**).

The ANSI control sequence occurs when the user presses a program function key on the keyboard or if the user enters an explicit control sequence.

The data consists of the parsed control sequence information. The Final Character is the valid or invalid final character. The Count indicates the number of parameters in the control sequence, with a maximum count of 10. These fields are followed by the Parameters. The *data* is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 2			
Final Character			
Count			
Parameter [1]			
:			
Parameter [Count]			

3 Locator

This event indicates the user has moved the locator or pressed a button on the locator.

The data consists of locator position and status information. The X value and the Y value field contain a relative movement (delta x, delta y) for a mouse and an absolute position (x, y) for a tablet. The Timestamp, which is elapsed time since system startup (IPL), is in sixtieths of a second.

The Buttons field contains the locator button status. For a mouse, each bit corresponds to a button, the most significant bit representing Button 1. A bit set to 1 indicates that the corresponding button is pressed. For a tablet, the most significant five bits represent the button pressed, according to the following scheme:

Status	**Button**
0	None pressed
1	Cursor upper left, stylus tip
2	Cursor upper right
3	Cursor lower left
4	Cursor lower right

For a tablet, the sixth most significant bit of the Buttons field indicates that the sensor is on (bit set) or off (bit not set).

The Type field contains a 0 if the locator is a mouse and a 1 if the locator is a tablet. The *data* is structured as follows:

<table>
<tr><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th></tr>
<tr><td colspan="4">Event type = 3</td></tr>
<tr><td colspan="4">X value</td></tr>
<tr><td colspan="4">Y value</td></tr>
<tr><td colspan="4">Type</td></tr>
<tr><td colspan="4">Buttons</td></tr>
<tr><td colspan="4">Timestamp</td></tr>
</table>

4 LPFK

This event type occurs when the user presses a key on the LPFK.

The data consists of the LPFK information. The LPFK field contains the decimal number of the LPFK pressed by the user, that is, 0 through 31. The Timestamp (time since system startup) is in sixtieths of a second. The *data* is structured as follows:

<table>
<tr><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th></tr>
<tr><td colspan="4">Event type = 4</td></tr>
<tr><td colspan="4">LPFK</td></tr>
<tr><td colspan="4">Timestamp</td></tr>
</table>

5 valuator

This event type occurs when the user turns a valuator dial.

The data consists of the valuator information. The Valuator field contains the decimal number, 0 through 7, of the valuator turned by the user. The Valuator Delta field contains the difference between the current valuator value and the last valuator value. The delta for a full turn is 256 for the IBM Valuator. The delta is positive for clockwise rotation and negative for counterclockwise rotation. The Timestamp (time since system startup) is in sixtieths of a second. The *data* is structured as follows:

<table>
<tr><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th></tr>
<tr><td colspan="4">Event type = 5</td></tr>
<tr><td colspan="4">Valuator</td></tr>
<tr><td colspan="4">Valuator Delta</td></tr>
<tr><td colspan="4">Timestamp</td></tr>
</table>

6 key code

This event type occurs when the virtual terminal is in non-translated mode *and* a keyboard key is pressed, held down, or released. The *data* is structured as follows:

<table>
<tr><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th></tr>
<tr><td colspan="4">Event type = 6</td></tr>
<tr><td colspan="4">Key Position Code</td></tr>
<tr><td colspan="4">Key Scan Code</td></tr>
<tr><td colspan="4">Status</td></tr>
</table>

Key position codes are found under “keyboard” on page 6-78. Status bits are found under “Input” on page 6-56.

7 pick event

This event type occurs while the pick operation is enabled and graphics primitives are being sent to the adapter. The *data* is structured as follows:

<table>
<tr><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th></tr>
<tr><td colspan="4">Event type = 7</td></tr>
<tr><td colspan="4">Pick Count</td></tr>
<tr><td colspan="4">Pick Center x</td></tr>
<tr><td colspan="4">Pick Center y</td></tr>
</table>

A pick event code is generated when a structure traversal occurs. The pick occurs when pixels are determined to intersect the pick window (defined by the pick enable window size). The detection mode is always immediate, so that an event is generated as soon as an event occurs. The pick event type is provided only for use with the IBM 5081 Display Adapter, and not for use with other displays.

10 Timeout

No data is returned.

It is important to note that **gsevwt** does not detect ANSI escape sequences. However, with the default virtual terminal keyboard mapping, it is not possible to generate an escape sequence by pressing a single key. Because **gsevwt** does parse ANSI control sequences, the routine cannot consider the press of the escape key an event, so the routine waits for the next character to decide if the escape implies the start of a control sequence. Only if the next character is not the left bracket does **gsevwt** return the escape and the next character.

If the return code indicates overflow, the most recent input events from enabled devices are lost.

Return Value

GS_SUCC	Successful.
GS_ROVR	Ring buffer overflow.
GS_UDRG	Ring undefined.
GS_PARM	Too many control sequence parameters.
GS_ICTL	Invalid final character.

Related Information

In this book: "keyboard" on page 6-78 and "Input" on page 6-56.

gsfatt

Purpose

Sets the fill attributes.

C Syntax

int gsfatt_ (*color*, *pattern*, *reserved***)**

**int ****color*, ******pattern*, ******reserved*;

FORTRAN Syntax

INTEGER function gsfatt (*color*, *pattern*, *reserved***)**

INTEGER *color*, *pattern*, *reserved*

Pascal Syntax

FUNCTION gsfatt_ (

VAR *color*, *pattern*, *reserved*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfatt** subroutine defines the attributes for the class of fill functions, which includes **gsfci**, **gsfell**, **gsfrec**, and **gsfply**.

Parameters

color Refers to an entry in the color map. If *color* is -1, the attribute is unchanged. The default color after initialization is 15.

pattern Contains a value from the following list:

Value	Display	Printer or Plotter
-1	No change	No change
0	Solid	Solid
1	Horizontal lines	Narrow right diagonal lines
2	Vertical lines	Medium right diagonal lines
3	135-degree lines	Wide right diagonal lines
4	45-degree lines	Narrow diagonal cross-hatched
5	Cross-hatched (horizontal and vertical lines)	Medium diagonal cross-hatched
6	Cross-hatched (45- and 135-degree lines)	Wide diagonal cross-hatched

The default pattern is solid (0).

Some printers and plotters support additional fill patterns that can be selected with a *pattern* index greater than 6. If the device you are using does not support additional fill patterns and you specify a *pattern* index greater than 6, then the **gsfatt** subroutine returns the value **GS_SYLI**.

The fill pattern does not meet the border of the filled area on printers and plotters.

reserved Represents a parameter that **gsfatt** ignores.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_SYLI	Invalid style index.

gsfci

Purpose

Fills a circle.

C Syntax

int gsfci_ (*cx*, *cy*, *cr***)**

int **cx*, **cy*, **cr*;

FORTRAN Syntax

INTEGER function gsfci (*cx*, *cy*, *cr***)**

INTEGER *cx*, *cy*, *cr*

Pascal Syntax

FUNCTION gsfci_ (

VAR *cx*, *cy*, *cr* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfci** subroutine fills a circle of the specified radius. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

cx, cy Define the coordinates of the center of the circle.

cr Defines the radius of the circle.

If the radius is zero, a single point is filled at the center.

If the radius is zero, a single point is filled at the center.

The fill pattern does not meet the border of the filled area on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_RDUS	Invalid radius specification.
GS_INAC	Virtual terminal inactive.

gsfell

Purpose

Fills an ellipse.

C Syntax

int gsfell_ (*cx*, *cy*, *ma*, *mi*, *ang***)**

**int ****cx*, ******cy*, ******ma*, ******mi*, ******ang*;

FORTRAN Syntax

INTEGER function gsfell (*cx*, *cy*, *ma*, *mi*, *ang***)**

INTEGER *cx*, *cy*, *ma*, *mi*, *ang*

Pascal Syntax

FUNCTION gsfell_ (

VAR *cx*, *cy*, *ma*, *mi*, *ang* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfell** subroutine fills an ellipse of the specified axes and angle. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsfell** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

cx, cy	Define the coordinates of the center of the ellipse.
ma, mi	Define half of the non-zero major and minor axes of the ellipse.
ang	Defines the angle between the major axis and the x-axis. If it is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is defined in tenths of degrees, from 0 to 3600, specified in a counterclockwise direction.

The fill pattern does not meet the border of the filled area on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_ELMM	Invalid major or minor axis.
GS_INAC	Virtual terminal inactive.
GS_ANGL	Invalid angle.
GS_NMEM	Insufficient resources.

gsfply

Purpose

Draws a filled polygon.

C Syntax

int gsfply_ (*number*, *x*, *y***)**

**int ****number*, ******x*, ******y*;

FORTRAN Syntax

INTEGER function gsfply (*number*, *x*, *y***)**

INTEGER *number*
INTEGER *x* **(*)**
INTEGER *y* **(*)**

Pascal Syntax

FUNCTION gsfply_ (

VAR *number*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfply** subroutine fills an area that is described by the points defined in the *number* and *x, y* parameters, with the color determined by the last call to the **gsfatt** subroutine.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

number	Defines the number of points in the coordinate arrays. This value must be 3 or more.
x, y	Define, as coordinate arrays, the points surrounding the polygon to fill.

The edges are treated as part of the area to be filled.

The **gsfply** subroutine fills a closed polygon with a pattern, generated by creating an edge between the first and the last points. The first and the last points described by the parameters may be equal, but it is not required and is actually less efficient.

The fill pattern does not meet the border of the filled area on printers and plotters.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; that is, the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.
GS_NMEM	Insufficient resources.
GS_INAC	Virtual terminal inactive.

gsfrec

Purpose

Draws a filled rectangle.

C Syntax

int gsfrec_ (*x1***,** *y1***,** *x2***,** *y2***)**

int ****x1*, ****y1***, ****x2***, ****y2***;**

FORTRAN Syntax

INTEGER function gsfrec (*x1***,** *y1***,** *x2***,** *y2***)**

INTEGER *x1***,** *y1***,** *x2***,** *y2*

Pascal Syntax

FUNCTION gsfrec_ (

VAR *x1***,** *y1***,** *x2***,** *y2***: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsfrec** subroutine fills the rectangular area defined by the lower leftmost and upper rightmost coordinate parameters, with the color determined by the last call to the **gsfatt** subroutine.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

x1, y1	Define the lower left corner of the rectangular area to fill.
x2, y2	Define the upper right corner of the rectangular area to fill.

The edges of the rectangle are treated as part of the area to be filled.

The fill pattern does not meet the border of the filled area on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_INAC	Virtual terminal inactive.

gsgtat

Purpose

Sets the attributes for the geometric text drawing functions.

C Syntax

```
int gsgtat_ (color, baseline, pre, expan, spac, height,
             upvectx, upvecty, alignhz, alignvt, font_ID, font)

int *color, *baseline, *pre, *expan, *spac, *height,
int *upvectx, *upvecty, *alignhz, *alignvt, *font_ID;
char *font;
```

FORTRAN Syntax

```
INTEGER function gsgtat (color, baseline, pre, expan, spac, height,
upvectx, upvecty, alignhz, alignvt, font_ID, font)

INTEGER color, baseline, pre, expan, spac, height
INTEGER upvectx, upvecty, alignhz, alignvt, font_ID
CHARACTER*n font
```

Pascal Syntax

```
FUNCTION gsgtat_ (

VAR color, baseline, pre, expan, spac, height: INTEGER;
VAR upvectx, upvecty, alignhz, alignvt, font_ID: INTEGER;
VAR font: ARRAY [0..k] of CHAR
): INTEGER [PUBLIC];
```

Description

The **gsgtat** subroutine defines the attributes and fonts for the geometric text drawing functions.

Note: The attributes defined by this command are applicable only to geometric text.

Parameters

color Specifies an entry in the color map for text color. If it is -1, the attribute is unchanged.

baseline Determines the direction of the geometric text drawing. The valid values are:

-1 Attribute remains unchanged.

0 Specifies 0 degrees, or left to right in the viewer's terms.

1 Specifies 90 degrees, or up in the viewer's terms.

2 Specifies 180 degrees, or right to left in the viewer's terms.

Note: The characters appear upside down.

3 Specifies 270 degrees, or down in the viewer's terms.

Note: The *baseline* parameter does not change character rotation. Use the *upvectx* and *upvecty* parameters to rotate text.

pre Specifies the desired text precision used in drawing text primitives. The valid values are:

-1 Attribute remains unchanged.

1 Character precision

2 Stroke precision.

expan Defines as a 32-bit fractional integer the deviation of the width/height ratio of the character from the ratio defined in the font. The expansion factor only changes the width of the character.

16	1	7	8
0 ——— 0	S	INTEGER	FRACTION

In the above figure, the first 16 bits contain zeros, S represents the sign bit, INTEGER represents the integer portion of the width/height ratio, and FRACTION represents the fractional portion of the ratio. A 32-bit integer value of -1 indicates that this attribute is unchanged.

spac Specifies the character spacing, or additional number of pixels to be inserted between characters. The value is a 16-bit signed integer. The preferred value for this parameter varies, based on the display in use. The maximum value that is allowed is equal to the display width in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.

height Specifies the current character height for geometric text in pixels. This value is defined as a 16-bit signed integer, with the maximum value equal to the height of the display in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.

upvectx, upvecty Specify the x and y coordinates for the up direction of a character or text string. The valid range for these values is ± the display dimensions in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.

The up vector is a two-dimensional vector on the text plane, specified by the current text draw. (The origin of the vector is defined by the geometric text command, **gsgtxt**.) Only the direction, not the length, of the vector is relevant.

alignhz Specifies the horizontal alignment of the text for subsequent text drawing. Values are as follows:

-1	Attribute is unchanged
1	Normal
2	Left
3	Center
4	Right

alignvt Specifies the vertical alignment of the text for subsequent text drawing. Values are as follows:

-1	Attribute is unchanged
1	Normal
2	Top
3	Cap
4	Half
5	Base
6	Bottom

font_ID Specifies the ID of the font as a 32-bit integer, which defines the type of font to use. This ID is determined by the user while defining each geometric font. Possible values are:

-1 A **font_ID** has been defined in a previous call to the **gsgtat** subroutine, and this attribute is unchanged.

1025 to 32767	These values are used to specify 1-byte geometric fonts, and refer to a value defined in each geometric font file.
32768 to 65535	These values are used to specify 2-byte geometric fonts, and refer to a value defined in each geometric font file.

Only 1 *font_ID* is active at any time. To change the **font_ID**, **gsgtat** must be called again with new *font_ID* and *font* parameters. When a new *font_ID* is specified, the previous *font_ID* is purged from the font table.

For 2-byte geometric text, up to 128 segment IDs can be used per *font_ID*. If the *font_ID* is the same as previously loaded, the current segment ID is added to the font tables.

When used with the *font* parameter, the *font_ID* is associated with the *font* used for font selection.

font Contains the null-terminated full path name of the file used when the font attribute is specified as user. If a **font_ID** is defined, this parameter must also be defined. A value of -1 for this parameter indicates that the attribute is unchanged. For information on the format of font files for geometric text, see "Geometric Text Fonts" on page 4-72.4.

Attributes are only valid for the currently active font.

This subroutine must be called before the **gsgtxt** subroutine or an error results.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_PREC	Invalid text precision value.
GS_EXPN	Invalid character expansion factor.
GS_FNTN	Invalid file name.
GS_INSV	Invalid spacing value.
GS_BASL	Invalid baseline direction.
GS_HIGH	Invalid height value.
GS_UPVT	Invalid up vector value.
GS_ALGN	Invalid alignment value.

Related Information

In this book: "fonts" on page 4-68, "Geometric Text Fonts" on page 4-72.4, and "gsgtxt" on page 7-78.

gsgtxt

Purpose

Writes geometric text.

C Syntax

int gsgtxt_ (*x*, *y*, *number*, *text***)**

int **x*, **y*, **number*;
char **text*;

FORTRAN Syntax

INTEGER function gsgtxt (*x*, *y*, *number*, *text***)**

INTEGER *x*, *y*, *number*
CHARACTER*n *text*

Pascal Syntax

FUNCTION gsgtxt_ (

VAR *x*, *y*, *number*: **INTEGER;**
VAR *text*: **ARRAY [1..***k***] of CHAR**
): INTEGER [PUBLIC];

Description

The **gsgtxt** subroutine writes geometric characters starting at the baseline position defined by the parameters and writes the number of characters indicated by the parameters according to the relevant attributes.

The relevant attributes are:

- Color map
- Plane mask
- Font
- Text color index
- Character expansion factor

- Character spacing
- Character height
- Character up vector
- Character alignment
- Baseline direction.

Parameters

x, y	Define the coordinates of the baseline position for writing geometric text.
number	Indicates the number of bytes to write from the *text* string. The maximum number of characters allowed is 1024 for single byte fonts and 512 for two-byte fonts, which is determined by the display and font in use.
text	Contains the N-bit ASCII codes for the characters to write, as an array.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_FBUF	Frame buffer overflow.
GS_INAC	Virtual terminal inactive.
GS_NOFT	Font not loaded.

Related Information

In this book: "fonts" on page 4-68 , "Geometric Text Fonts" on page 4-72.4, "gsgtat" on page 7-73, and "gsqgtx" on page 7-119.

gsinit

Purpose

Initializes the GSL subroutines.

C Syntax

int gsinit_ (*buffer*, *size*, *save_restore*, *f_grant*, *f_retract*, *fildes***)**

int **buffer*, **size*, **save_restore*;
int (**f_grant***) ( ), (****f_retract***) ( );**
int **fildes*;

FORTRAN Syntax

INTEGER function gsinit (*buffer*, *size*, *save_restore*, *f_grant*, *f_retract*, *fildes***)**

INTEGER *buffer* **(*)**, *size*, *save_restore*, *fildes*
EXTERNAL *f_grant*, *f_retract*

Pascal Syntax

FUNCTION gsinit_ (

VAR *buffer*: **ARRAY [0..***k***] of INTEGER;**
VAR *size*, *save_restore*, *f_grant*, *f_retract*, *fildes*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsinit** subroutine initializes the GSL. It allocates any private storage required, and sets attributes to the default values where necessary. It also forces the virtual terminal of the application to Monitor Mode and sets up the signal processing routines for the **SIGRETRACT** and **SIGGRANT** signals, and optionally, the **SIGMSG** signal.

Parameters

buffer Defines the Monitor Mode input ring buffer to be used by the GSL input functions. *buffer* must be word aligned and at least 128 bytes long. For output to a printer or plotter device, set the *buffer* parameter to -1. (In C, *buffer* is a pointer to an integer containing the value -1. In Pascal, it is a variable containing the value -1.)

size Defines the length of *buffer* in bytes. Depending on the value of *size*, **gsinit** performs the following actions:

- *size*=0 The GSL ignores the *buffer* parameter and does not provide input support. The application must provide a means for receiving input events and can use the **read** system call or set up its own ring buffer mechanism.

  The IBM 5081 Display Adapter requires a ring buffer for input events. If you do not define a buffer (that is, if *size*=0 and *buffer* is not defined), the GSL defines a buffer to be used only by GSL for the IBM 5081 Display Adapter. If you define a ring buffer after this point, the IBM 5081 Display Adapter GSL will not work.

- *size*<128 The **gsinit** subroutine does not initialize the GSL.

- *size*≥128 The GSL establishes the virtual terminal linkage to the input ring buffer provided by the application and provides input support and sets up a **SIGMSG** signal catcher.

save_restore Determines whether to save the display frame buffer and adapter states.

If *save_restore* is non-zero, the GSL saves the current contents of the display frame buffer as well as the current adapter state when the virtual terminal must become inactive and restores both the frame buffer contents and adapter state when it becomes active.

If *save_restore* is zero, the GSL saves only the adapter state and assumes that the application either saves the frame buffer or reconstructs it in some fashion.

f_grant Sets up processing of the **SIGGRANT** signal. If *f_grant* is non-zero, it is assumed to be the address of an application supplied function, and the GSL calls the function as part of the **SIGGRANT** signal handling. If *save_restore* is non-zero, the application function is called before the frame buffer is restored.

f_retract Sets up processing of the **SIGRETRACT** signal. If *f_retract* is non-zero, it is assumed to be the address of an application supplied function, and the GSL calls the function as part of the **SIGRETRACT** signal handling.

fildes	Determines where output is directed. The output device is specified by one of the following: • The value -1, which specifies standard output. • A file descriptor returned by a **creat**, **open**, **dup**, or **fcntl** system call. • A null-terminated character string up to 11 characters long, which names an environment variable defining a printer or plotter device. In this case, the value of the *buffer* parameter must be -1. (See "Printers and Plotters" on page 7-11.) (In C, *fildes* is a pointer to a file descriptor, an integer, or a character string. In Pascal, it is a variable containing one of these values.)

If the initialization process is unsuccessful, the virtual terminal is not placed in Monitor Mode and invocation of any other GSL routines will cause unpredictable results.

For printers or plotters, if initialization is unsuccessful, the application can either terminate or re-drive the initialize function with a valid character string as a means of correcting the problem.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; that is, the *k* in the routine declaration must be a constant.

Pascal cannot directly provide the address of a routine. An assembler function may be used to derive the address of a routine passed to the GSL.

The *f_grant* and *f_retract* routines supplied by the application are called on the signal level and *must return.* These application routines must not use either **setjmp** or **longjmp** subroutines.

The GSL supports use of the **sdb** symbolic debugger by redirection to a supplied file descriptor. If two virtual terminals are open and the GSL application runs on one, the application may get the file descriptor for the second and supply that descriptor at GSL initialization. The GSL directs its output to the second virtual terminal while **sdb** directs its output to the first; either is activated in the standard manner.

The user routine called at **SIGGRANT** can be called before **gsinit** returns to the application.

Return Value

GS_SUCC	Successful.
GS_HBUS	Cannot access hardware bus.
GS_ADPT	Invalid display type.
GS_FONT	Cannot access default font.
GS_RING	Buffer too small.
GS_HDCP	Invalid file descriptor for hard copy output.

GS_HDLK	Unable to create lock file.
GS_HDIM	Insufficient memory.
GS_HDDB	Device is busy.
GS_HDNA	Physical device not attached.
GS_HDMG	Maximum number of graphics devices open.
GS_HDIF	No system inter process communication buffers left.
GS_HDSF	The **fork** system call failed.
GS_HDGO	Specified graphics device already open.
GS_HDGN	Specified graphics device does not exist.
GS_HDGU	Specified graphics device driver is unknown.

Related Information

In this book: "Printers and Plotters" on page 7-11.

gslatt

Purpose

Sets the line attributes.

C Syntax

int gslatt_ (*color*, *style***)**

int **color*, **style*;

FORTRAN Syntax

INTEGER function gslatt (*color*, *style***)**

INTEGER *color*, *style*

Pascal Syntax

FUNCTION gslatt_ (

VAR *color*, *style*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gslatt** subroutine defines the attributes for the class of line drawing functions.

Parameters

color Refers to a line color entry in the color map. If it is -1, the attribute is unchanged. The default color is 15.

style Sets or resets the line style pattern. The line *style* may be one of the following:

Value	**Display**	**Printer or Plotter**
-1	No change	No change
0	Solid	Solid
1	Dash	Dash
2	Dot	Dot
3	Dash-dot	Dash-dot
4	Dash-dot-dot	Dash-dot-dot
100	Continuous solid	Solid
101	Continuous dash	Dash
102	Continuous dot	Dot
103	Continuous dash-dot	Dash-dot
104	Continuous dash-dot-dot	Dash-dot-dot
150	Continuous user-supplied	Not available

The default style is solid (0).

The GSL supplied line style patterns are implemented in a device-dependent fashion. All line style indices not described above are reserved.

For line styles 1-99, the GSL line drawing functions ensure that a line or line segment starts and ends with a run of the line color. For example, the GSL *does not* continue the pattern from one polyline segment to another.

For line styles 100-150, the GSL continues the pattern across multiple lines or line segments until the application makes another call to **gslatt** to reset the line pattern. In this case, unlike styles 1-99, the GSL *does* continue the pattern from one polyline segment to another. Continuous line styles are not available on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_SYLI	Invalid style index.

gslcat

Purpose

Sets the locator attributes.

C Syntax

int gslcat_ (*hg*, *vg*)

**int ****hg*, ******vg*;

FORTRAN Syntax

INTEGER function gslcat (*hg*, *vg*)

INTEGER *hg*, *vg*

Pascal Syntax

FUNCTION gslcat_ (

VAR *hg*, *vg*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gslcat** subroutine sets the locator attributes. Its effect depends on the type of locator attached. For a mouse, **gslcat** sets the thresholds. For a tablet, it sets the dead zone.

Parameters

hg, vg	Define the horizontal and vertical values for the locator threshold or dead zone, in units of 0.25 millimeter.

The mouse ***thresholds*** determine the granularity of input events reported, or the amount of horizontal or vertical mouse movement required before an event occurs.

The tablet ***dead zone*** is an area of the tablet in which no event reports occur, even if the tablet sensor is present. This dead zone allows the application to make the tablet aspect ratio compatible with the display and allows tablets of different sizes to appear the same size to an application. The dead zone acts as a border around the tablet. The device driver reports movement only when the x value is greater than or equal to *hg* or less than or equal to (maximum tablet value - *hg*), and the y value is greater than or equal to *vg* or less than or equal to (maximum tablet value - *vg*).

An attempt to set the locator attributes may fail for a variety of reasons, the most likely of which is that the device is not attached. The nature of the problem can be determined with a specific **ioctl** to the virtual terminal. (See "hft" on page 6-23 for more information.)

Note that the **gslcat** subroutine allows an application to set the mouse thresholds or the tablet dead zone such that no events occur even if the device is enabled.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: "hft" on page 6-23.

gsline

Purpose

Draws a line between two points.

C Syntax

int gsline_ (*x1*, *y1*, *x2*, *y2***)**

**int ****x1*, ******y1*, ******x2*, ******y2*;

FORTRAN Syntax

INTEGER function gsline (*x1*, *y1*, *x2*, *y2***)**

INTEGER *x1*, *y1*, *x2*, *y2*

Pascal Syntax

FUNCTION gsline_ (

VAR *x1*, *y1*, *x2*, *y2*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsline** subroutine draws a line, as defined by the current relevant attributes, from the first point to the second point defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

x1, y1	Defines the coordinates of one end point of the line drawn by **gsline**.
x2, y2	Defines the coordinates of the second point of the line drawn by **gsline**.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_INAC	Virtual terminal inactive.

gslock

Purpose

Postpones signal processing.

C Syntax

int gslock_ ( )

FORTRAN Syntax

INTEGER function gslock ( )

Pascal Syntax

FUNCTION gslock_ ( ): INTEGER [PUBLIC];

Description

The **gslock** subroutine causes the GSL not to acknowledge the **SIGRETRACT** signal, if it occurs, until the application requests resumption of the signal handling with the **gsunlk** subroutine. This permits the application to access the display frame buffer directly.

If the virtual terminal is inactive when the application calls **gslock** and the GSL has been instructed to save the frame buffer when the virtual terminal becomes inactive, **gslock** suspends the application until the virtual terminal becomes active and then returns a successful return code. If the GSL has been instructed not to save the frame buffer, **gslock** returns the **GS_INAC** return code immediately. The application must not access the display frame when **GS_INAC** is returned.

Note: If **SIGRETRACT** signal processing is suspended for more than 30 seconds, it is possible that a generated **SIGRETRACT** signal may be suspended long enough for the **SIGKILL** signal to occur, terminating the application process.

Return Value

GS_SUCC	Virtual terminal active, safe to write to frame buffer.
GS_INAC	Virtual terminal inactive.

gslop

Purpose

Specifies the logical operation used when drawing lines.

C Syntax

int gslop_ (*operation***)**

int **operation*;

FORTRAN Syntax

INTEGER function gslop (*operation***)**

INTEGER *operation*

Pascal Syntax

FUNCTION gslop_ (

VAR *operation* **INTEGER;**
): INTEGER [PUBLIC];

Description

The **gslop** subroutine specifies the logical operation used for drawing the GSL line-oriented, fill, save/restore, and polymarker primitives. It does not apply to the text primitives.

Parameters

operation Indicates the logical operation to perform between the primitive being drawn and the current contents of the frame buffer.

In the following table, please note:

- The source pixels represent bits of data to be merged in some way with the corresponding bits of data in the destination rectangle.

- The first three columns of the table specify the operations you can perform, and the **Code** column contains the corresponding value you should specify for the *operation* parameter.
- A ~ (tilde) represents the logical INVERSE.

Type of Source	Logical Operation	Type of Destination	Code
		Destination clear	0
		Set Destination	15
	No operation	Destination	5
		~Destination	10
Source	REPLACE	Destination	3
Source	AND	Destination	1
Source	AND	~Destination	2
Source	EXCLUSIVE-OR	Destination	6
Source	OR	Destination	7
Source	OR	~Destination	11
~Source	REPLACE	Destination	12
~Source	AND	Destination	4
~Source	AND	~Destination	8
~Source	EXCLUSIVE-OR	Destination	9
~Source	OR	Destination	13
~Source	OR	~Destination	14

Replace (3) is the default logical operation.

Currently, the GSL provides only replace and exclusive-or (codes 3 and 6 respectively) for displays other than the IBM 5081 Display. The full set of logical operations is supported for the IBM 5081 Display.

For printers and plotters, the operations performed are the same as those for displays, except that a value of 0 turns the color off, and a value of 15 changes the color to white.

The GSL performs each of the boolean operations for each bit of the source and destination color values enabled by the plane mask. The destination receives the color value that results from the operation.

The logical operations are performed on the color index rather than the color itself. This can cause some operations on color displays to produce results that are not expected.

Return Value

GS_SUCC	Successful.
GS_LONS	Logical operation not supported.

gslpat

Purpose

Sets the LPFK indicators.

C Syntax

int gslpat_ (*indicators*)

int ****indicators*;**

FORTRAN Syntax

INTEGER function gslpat (*indicators*)

INTEGER *indicators*

Pascal Syntax

FUNCTION gslpat_ (

VAR *indicators*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gslpat** subroutine turns on or off the indicators on the Lighted Program Function Keyboard.

Parameters

indicators Specifies the state of the LPFK indicators. Each bit of *indicators* corresponds to an indicator on the LPFK, with the most significant bit setting the desired state (1 = on, 0 = off) for the indicator for LPFK 0, the next most significant bit setting the state for the indicator for LPFK 1, and so on.

The default state for all indicators is off.

An attempt to set the LPFK indicators may fail for a variety of reasons, the most likely of which is that the device is not attached. The nature of the problem can be determined with a specific **ioctl** to the virtual terminal. (See "hft" on page 6-23 for more information.)

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: "hft" on page 6-23.

gsmask

Purpose

Defines planes to be modified.

C Syntax

int gsmask_ (*mask*)

**int ****mask*;

FORTRAN Syntax

INTEGER function gsmask (*mask*)

INTEGER *mask*

Pascal Syntax

FUNCTION gsmask_ (

VAR *mask*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmask** subroutine defines the planes actually modified by the line, text, and fill functions.

Parameters

mask	Indicates which planes of the display adapter frame buffer can be modified by the output functions. The most significant bits of the input are used to set the plane mask.

Return Value

GS_SUCC	Successful.
GS_INAC	Virtual terminal inactive.

gsmatt

Purpose

Sets the polymarker attribute.

C Syntax

int gsmatt_ (*color*, *style*, *width*, *height*, *pattern*, *0x*, *0y***)**

int **color*, **style*, **width*, **height*, **pattern*, **0x*, **0y*;

FORTRAN Syntax

INTEGER function gsmatt (*color*, *style*, *width*, *height*, *pattern*, *0x*, *0y***)**

INTEGER *color*, *width*, *height*, *pattern*, *0x*, *0y*

Pascal Syntax

FUNCTION gsmatt_ (

VAR *color*, *style*, *width*, *height*: **INTEGER;**
pattern: **ARRAY** [1..*k*] **of INTEGER;**
0x, *0y*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmatt** subroutine defines the marker for the GSL.

Parameters

color	Refers to a marker color entry in the color map. If it is -1, the attribute is unchanged. The default value for color is 7, white.

style Defines the polymarker *style* as one of the following:

Value	Display	Printer or Plotter
-1	No change	No Change
0	User-defined (by *width*, *height*, *pattern*, *0x*, *0y*)	Not available
1	Dot (filled circle)	Point
2	Plus (+)	Plus (+)
3	Asterisk (*)	Asterisk (*)
4	Circular shape	Square shape
5	Cross (×)	Cross (×)
6	Unfilled box	Diamond

width, height Define in pixels the width and the height of the bit pattern to be used as the marker. If *width* or *height* equals -1, then the pattern remains unchanged.

pattern Defines the image used as a marker. The ceiling of (*width* / 32) indicates the number of words per row and *height* indicates the number of rows. The marker data must be supplied in row (scan line) major order. If *width* implies partial use of a word, the rest of the word is unused. To fully define the marker pattern, *pattern* should be ((ceiling(*width* / 32)) × *height*) words in length.

0x, 0y Indicate the coordinates of the origin of the marker relative to the lower leftmost corner (0, 0) of the marker pattern. The origin must be placed inside the marker pattern, so that $0x < width$ and $0y < height$. The origin of the marker is placed at the position indicated when the application places a marker with the **gsplym** subroutine. (See "gsplym" on page 7-108.) If *0x* equals -1, then the origin remains unchanged.

The maximum size of the marker is device dependent. It equals the height and width of the display, which may be determined by calling the **gsqdsp** subroutine.

Note: The GSL subroutines do not make a copy of a user-defined polymarker. Changes or reuse of the storage where a user-defined shape is in use can cause unpredictable results.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_PMSZ	Marker size invalid.
GS_PMOR	Marker origin invalid.
GS_PMSY	Marker style invalid.

Related Information

In this book: "gsplym" on page 7-108.

gsmcur

Purpose

Moves the cursor and makes it visible.

C Syntax

int gsmcur_ (*x*, *y*)

**int ****x*, ******y*;

FORTRAN Syntax

INTEGER function gsmcur (*x*, *y*)

INTEGER *x*, *y*

Pascal Syntax

FUNCTION gsmcur_ (

VAR *x*, *y*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmcur** subroutine makes the cursor visible (if not already visible) and positions the cursor origin at the point indicated by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Cursor pattern
- Cursor color index
- Cursor origin.

Parameters

x, *y*	Indicate the coordinates of the desired position of the cursor origin.

The cursor attributes *must* be set with the **gscatt** subroutine before calling **gsmcur** (see "gscatt" on page 7-24).

The cursor is non-destructive. This is achieved in a device-dependent manner.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_UCUR	Undefined cursor.
GS_INAC	Virtual terminal inactive.

Related Information

In this book: "gscatt" on page 7-24.

gsmult

Purpose

Draws a multiline, a set of lines that connect alternate pairs of points in a sequence.

C Syntax

int gsmult_ (*number*, *x*, *y***)**

int **number*, **x*, **y*;

FORTRAN Syntax

INTEGER function gsmult (*number*, *x*, *y***)**

INTEGER *number*, *x* **(*)**, *y* **(*)**

Pascal Syntax

FUNCTION gsmult_ (

VAR *number*: **INTEGER;**
VAR *x*, *y*: **ARRAY** [1..*k*] **of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsmult** subroutine draws lines, as defined by the current relevant attributes, between alternate pair of points defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

number	Defines the number of points in the coordinate arrays. It must be a multiple of 2, with 2 as the minimum value.
x, *y*	Define the points for line drawing.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.
GS_INAC	Virtual terminal inactive.

gspcls

Purpose

Defines the end of a shape to fill.

C Syntax

int gspcls_ ( )

FORTRAN Syntax

INTEGER function gspcls

Pascal Syntax

FUNCTION gspcls_ : INTEGER [PUBLIC];

Description

The **gspcls** subroutine defines the end of a particular two dimensional shape to be filled, then fills the shape.

See “gsbply” on page 7-20 and “gseply” on page 7-50 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: “gsbply” on page 7-20 and “gseply” on page 7-50.

gsplym

Purpose

Draws a polymarker, a marker at each of a set of specified points.

C Syntax

int gsplym_ (*number*, *x*, *y***)**

**int ****number*, ***x*, ***y*;

FORTRAN Syntax

INTEGER function gsplym (*number*, *x*, *y***)**

INTEGER *number*, *x* **(*)**, *y* **(*)**

Pascal Syntax

FUNCTION gsplym_ (

VAR *number*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gsplym** subroutine places a marker, defined by the current relevant attributes, at each point defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Logical operation (except on IBM 5081 Display)
- Polymarker color index
- Polymarker style index.

Parameters

number	Defines the number of points in the coordinate arrays. It must be ≥ 1.
x, *y*	Define, as coordinate arrays, the location where the origin of each polymarker is placed.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.

gspoly

Purpose

Draws a polyline, a set of lines that connects a sequence of points.

C Syntax

int gspoly_ (*number*, *x*, *y***)**

**int ****number*, **x*, **y*;

FORTRAN Syntax

INTEGER function gspoly (*number*, *x*, *y***)**

INTEGER *number*, *x* **(*)**, *y* **(*)**

Pascal Syntax

FUNCTION gspoly_ (

VAR *number*: **INTEGER;**
VAR *x*, *y*: **ARRAY [1..***k***] of INTEGER**
): INTEGER [PUBLIC];

Description

The **gspoly** subroutine draws lines, as defined by the current relevant attributes, between each pair of points defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

number	Defines the number of points in the coordinate arrays. It must be ≥ 2.
x, *y*	Define the points for line drawing.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.
GS_INAC	Virtual terminal inactive.

gspp

Purpose

Sets plotter pen speed.

C Syntax

int gspp_ (*penspd*)

**int ****penspd*;

FORTRAN Syntax

INTEGER function gspp (*penspd*)

INTEGER *penspd*

Pascal Syntax

FUNCTION gspp_ (

VAR *penspd*: **INTEGER;**
): INTEGER [PUBLIC];

Description

The **gspp** subroutine sets the plotter pen speed.

Parameters

penspd Specifies the pen speed as a value from 0 to 100, giving a percentage of the maximum speed of the plotter. The initial pen speed is 100 percent.

Return Value

GS_SUCC	Successful.
GS_USUC	Invalid parameter value.

gsqdsp

Purpose

Returns characteristics of the display monitor and adapter.

C Syntax

void gsqdsp_ (*display***)**

**int ****display*;

FORTRAN Syntax

subroutine gsqdsp (*display***)**

INTEGER *display* **(32)**

Pascal Syntax

PROCEDURE gsqdsp_ (

VAR *display*: **ARRAY [1..32] of INTEGER
): INTEGER [PUBLIC];**

Description

The **gsqdsp** subroutine returns an array containing the display adapter and monitor characteristics.

Parameters

display Contains, on return, the relevant display/monitor characteristics. The following table describes the information in the array. Each entry is a word.

Entry Description (measure)

1 Display/monitor ID. For a printer or plotter, this value is " HC", right-justified in the word.
2 Displayed width of the frame buffer in pixels.
3 Displayed height of the frame buffer in pixels.
4 Physical width of display in millimeters.
5 Physical height of display in millimeters.
6 Number of bit planes or number of bits/pixel.
7 Adapter characteristic flags. (Bit 0 is the most significant bit.) Bits set these characteristics:

- 0 Color or monochrome; 0 = color, 1 = monochrome
- 1 By plane or by pixel; 0 = by plane, 1= by pixel (always 1 for printers and plotters).
- 2 Software or hardware cursor; 0 = software, 1 = hardware (always 0 for printers and plotters).
- 3-31 Reserved bits.

8 Number of bits for Red digital-to-analog converter (always 2 for printers and plotters).
9 Number of bits for Green digital-to-analog converter (always 2 for printers and plotters).
10 Number of bits for Blue digital-to-analog converter (always 2 for printers and plotters).
11 Minimum cursor width (pixels) (always 0 for printers and plotters).
12 Minimum cursor height (pixels) (always 0 for printers and plotters).
13 Maximum cursor width (pixels) (always 0 for printers and plotters).
14 Maximum cursor height (pixels) (always 0 for printers and plotters).
15 Color table size. For printers and plotters, this specifies the number of colors.
16 Font class:

- 1 Compressed (always 1 for printers and plotters).
- 2 Uncompressed.

17 Logical operation capability.

If the value is zero, the adapter supports all 16 two-operand logical operations and all 256 three-operand logical operations. If non-zero, the most significant bits represent the two-operand logical operations supported; bit 0 corresponds to logical operation 0, bit 1 to logical operation 1, and so on (see “gslop” on page 7-92).

18-32 Reserved.

Information from this query can be used to scale application coordinates to those of the frame buffer.

Even if the adapter supports **no** logical operations, the results of the query indicate that the adapter supports replace and exclusive-or (logical operations 3 and 6, respectively). The GSL emulates the latter, if necessary.

The following display adapter IDs are valid:

0x0402	IBM 6153 Display Adapter
0x0405	IBM 6155 Display Adapter
0x0406	IBM 6154 Display Adapter
0x0408	IBM 5081 Display Adapter.

Related Information

In this book: "gslop" on page 7-92.

gsqfnt

Purpose

Returns information about the current font.

C Syntax

void gsqfnt_ (*font*)

**int ****font*;

FORTRAN Syntax

subroutine gsqfnt (*font*)

INTEGER *font* **(32)**

Pascal Syntax

PROCEDURE gsqfnt_ (

VAR *font*: **ARRAY [1..32] of INTEGER
): INTEGER [PUBLIC];**

Description

The **gsqfnt** subroutine returns information about the active font.

Parameters

font Contains, on return, the characteristics of the current font. The following table describes the information in the array. Each entry is a word. Dimensions are in pixels and the origin is at the lower left corner of the character box.

Entry	Description
1	Class: 1 = compressed; 2 = uncompressed IBM 5081 Display format (always 1 for printers and plotters).
2	Font ID.
3	Style.
4	Attribute flags: bit 31 bold bit 30 italic bit 00 proportionally spaced. (This entry always has all bits set to 0 for printers and plotters.)
5	Number of characters. For printers and plotters, this is the number of fonts × 128.
6	Character baseline. For printers and plotters, no text alignment is allowed and this value is always -1.
7	Character capsline. For printers and plotters, no text alignment is allowed and this value is always -1.
8	Character width. For printers and plotters, the character width is given in pixels. For a proportionally spaced font, the width value represents the maximum width allowed.
9	Character height. For printers and plotters, the character height is given in pixels.
10	Underscore top line. For printers and plotters, underscoring is not available and this value is always -1.
11	Underscore bottom line. For printers and plotters, underscoring is not available and this value is always -1.
12-32	Reserved.

gsqgtx

Purpose

Returns information about the current geometric font.

C Syntax

void gsqgtx_ (*font*, *select*)

int *_font_, *_select_;

FORTRAN Syntax

subroutine gsqgtx (*font*, *select*)

INTEGER *font* **(32),** *select*

Pascal Syntax

PROCEDURE gsqgtx_ (

VAR *font*: **ARRAY [1..32] of INTEGER;**
select: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsqgtx** subroutine returns information about the active geometric font.

Parameters

font Contains, on return, the characteristics of the selected PCS descriptor header. The following table describes the information in the array. Each entry is a word. Dimensions are in pixels and the origin is at the lower left corner of the character box.

Entry	Description
1	Font ID.
2	Segment ID.
3	0 = EBCDIC; 1 = ASCII.
4	Range of **x** (P).
5	Range of **y** (Q).
6	Starting character code. Range is 0x21 to 0xFE.
7	Last character code. Range is 0x21 to 0xFE.
8	Font baseline. Value in pixels in the **y** direction.
9	Font capline. Value in pixels in the **x** direction.
10	Default error code point.
11-32	Reserved.

select Determines the type of query.

A value of -1 returns the following information in the *font* parameter buffer:

Word 1	Current active *font_ID*
Word 2	Number of PCS descriptor headers (segments for 2-byte text) loaded at the time of the query.

A value other than -1 returns the PCS descriptor header associated with that number in the table.

Related Information

In this book: "fonts" on page 4-68, "Geometric Text Fonts" on page 4-72.4, "gsgtat" on page 7-73, and "gsgtxt" on page 7-78.

gsqloc

Purpose

Returns information about the locator.

C Syntax

void gsqloc_ (*loc_type*, *x_res*, *y_res*, *hg*, *vg***)**

int ******loc_type*, ******x_res*, ******y_res*, ******hg*, ******vg*;

FORTRAN Syntax

subroutine gsqloc (*loc_type*, *resolution*, *hg*, *vg***)**

INTEGER *loc_type*, *resolution*, *hg*, *vg*

Pascal Syntax

PROCEDURE gsqloc_ (

VAR *loc_type*, *resolution*, *hg*, *vg*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsqloc** subroutine returns the type of the locator, the resolution of the device, and the current setting of the relative device thresholds or the absolute device dead zone values (see “gslcat” on page 7-86).

Parameters

loc_type Indicates the type of locator. If the most significant bit of *loc_type* is 0, the locator is a mouse. Otherwise, it is a tablet. For a tablet, the next most significant two bits are:

00	No sensor is attached.
01	A stylus is attached.
10	A four button puck is attached.

x_res, y_res Indicate the horizontal and vertical resolution of the device in millimeters per 100 counts.

hg, vg Define the horizontal and vertical values for the locator threshold or dead zone in units of 0.25 millimeters. (See “gslcat” on page 7-86.)

An attempt to get the locator attributes may fail for a variety of reasons, the most likely of which is that the device is not attached. The nature of the problem can be found via a specific **ioctl** to the virtual terminal. (See “hft” on page 6-23 for more information.)

Return Value

GS_SUCC	Successful.
GS_UNSC	Unsuccessful.

Related Information

In this book: “hft” on page 6-23 and “gslcat” on page 7-86.

gsrrst

Purpose

Restores a rectangular block.

C Syntax

int gsrrst_ (*buffer*, *x1*, *y1*, *x2*, *y2***)**

**int ****buffer*, ***x1*, ***y1*, ***x2*, ***y2*;

FORTRAN Syntax

INTEGER function gsrrst (*buffer*, *x1*, *y1*, *x2*, *y2***)**

INTEGER *buffer* **(*)**, *x1*, *y1*, *x2*, *y2*

Pascal Syntax

FUNCTION gsrrst_ (

VAR *buffer*: **ARRAY** [1..*k*] **of INTEGER;**
VAR *x1*, *y1*, *x2*, *y2*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsrrst** subroutine restores a block of pixels saved to the frame buffer by the **gsrsav** subroutine. (See "gsrsav" on page 7-125.)

The relevant attributes are:

- Plane mask
- Logical operation.

Parameters

buffer	Indicates where **gsrrst** should restore the block of pixels from.
x1, y1	Define the coordiantes of the lower left corner of the rectangular area to restore.
x2, y2	Define the coordinates of the upper right corner of the rectangular area to restore.

The intended purpose of the **gsrsav** and **gsrrst** subroutines is efficient saving and restoring of pixel blocks displayed temporarily at a fixed location in the frame buffer. Because the GSL saves the frame buffer contents in a device-dependent fashion, it is generally not possible to use **gsrsav** and **gsrrst** to correctly move blocks of pixels from one position to another in a plane oriented adapter, nor is it possible for the application to manipulate the *buffer* without careful consideration of adapter characteristics, block size, and position of the block in the frame buffer.

For further information on moving and storing blocks of pixels, see "gsxblt" on page 7-139.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_INAC	Virtual terminal inactive.

Related Information

In this book: "gsrsav" on page 7-125 and "gsxblt" on page 7-139.

gsrsav

Purpose

Saves a rectangular block.

C Syntax

int gsrsav_ (*buffer*, *x1*, *y1*, *x2*, *y2***)**

int **buffer*, **x1*, **y1*, **x2*, **y2*;

FORTRAN Syntax

INTEGER function gsrsav (*buffer*, *x1*, *y1*, *x2*, *y2***)**

INTEGER *buffer* **(*)**, *x1*, *y1*, *x2*, *y2*

Pascal Syntax

FUNCTION gsrsav_ (

VAR *buffer*: **ARRAY [1..***k***] of INTEGER;**
VAR *x1*, *y1*, *x2*, *y2*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsrsav** subroutine saves a block of pixels, defined by the input rectangle, in storage starting at the address indicated. This stored block can be restored with the **gsrrst** subroutine. (See "gsrrst" on page 7-123.)

The relevant attributes are:

- Plane mask
- Logical operation.

Parameters

buffer Indicates where **gsrsav** should save the block of pixels.

The size of the *buffer* depends on the size of the rectangle and on the device organization. For devices organized by plane, the plane mask attribute determines the number of planes saved for each pixel. For devices organized by pixel, the entire pixel is always saved. For both organizations, the unit of access to the frame buffer also plays a role in calculating the size of the *buffer*. See "gscmap" on page 7-32 for details.

Note that the **gsrsav** subroutine does not check whether the *buffer* is too small to contain the pixel block. Serious consequences can result if the *buffer* is too small. However, a *buffer* size equal to

```
(((y2-y1+1)/32+2)*(x2-x1+1))
```

will hold all save images.

x1, y1 Define the lower left corner of the rectangular area to save. That is, *x1* is the greatest lower bound of the pixels saved in *x*.

x2, y2 Define the upper right corner of the rectangular area to save. That is, *x2* is the least upper bound of the pixels saved in *x*.

The intended purpose of the **gsrsav** and **gsrrst** subroutines is efficient saving and restoring of pixel blocks displayed temporarily at a fixed location in the frame buffer. Because the GSL saves the frame buffer contents in a device-dependent fashion, it is generally not possible to correctly move blocks of pixels from one position to another in a plane-oriented adapter using **gsrsav** and **gsrrst**, nor is it possible to manipulate the *buffer* without careful consideration of adapter characteristics, block size, and position of the block in the frame buffer.

For further information on moving and storing blocks of pixels, see "gsxblt" on page 7-139.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_INAC	Virtual terminal inactive.

Related Information

In this book: "gscmap" on page 7-32 , "gsrrst" on page 7-123, and "gsxblt" on page 7-139.

gstatt

Purpose

Sets the text attributes for annotated text.

C Syntax

int gstatt_ (*color*, *page*, *baseline*, *font*, *name***)**

int **color*, **page*, **baseline*, **font*;
char *name;

FORTRAN Syntax

INTEGER function gstatt (*color*, *page*, *baseline*, *font*, *name***)**

INTEGER *color*, *page*, *baseline*, *font*
CHARACTER**n name*

Pascal Syntax

FUNCTION gstatt_ (

VAR *color*, *page*, *baseline*, *font*: **INTEGER;**
VAR *name*: **ARRAY [0..***k***] of CHAR**
): INTEGER [PUBLIC];

Description

The **gstatt** subroutine defines the attributes for the class of text drawing functions.

Parameters

color Specifies a text color entry in the color map. If it is -1, the attribute is unchanged.

page Specifies the code page of a font for the display to use. The valid values for IBM supplied fonts are 0, 1, and 2 for code pages P0, P1, and P2, respectively. The value -1 indicates no change.

For printers and plotters, the *page* parameter is a font value specification. The IBM 3812 Pageprinter supports 128 different code pages. Again, the value -1 indicates no change.

baseline Determines the direction of the text drawing. The valid values are:

-1 Attribute remains unchanged.

0 Specifies 0 degrees, or left to right in the viewer's terms.

1 Specifies 90 degrees, or up in the viewer's terms.

2 Specifies 180 degrees, or right to left in the viewer's terms.

Note: The characters appear upside down.

3 For 270 degrees, or down in the viewer's terms.

If the baseline is other than 0 degrees, the user has pre-rotated the fonts. When a baseline change is made, another font path name is required.

font Specifies, for displays, the font to use for text output operations. If the *font* index is -1, it remains unchanged. If *font* index is 0, then **gstatt** uses the font specified by the *name* parameter.

For printers and plotters, the *font* parameter specifies the vertical height of the font in pixels.

name Contains the null-terminated full path name of the file used when the font attribute is specified as user. See "fonts" on page 4-68 for the format of this file.

If a single-shift control is outstanding and **gstatt** is called to change the code page or the font, then the single-shift control is ignored. (See "Code Page Switching" on page 5-9 for details about single-shift controls.)

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_CPID	Invalid code page identifier.
GS_BASL	Invalid baseline direction.
GS_FNTI	Invalid font index.
GS_FNTN	Invalid file name.
GS_IVFF	Invalid font format.

Related Information

In this book: "fonts" on page 4-68 and "Code Page Switching" on page 5-9.

gsterm

Purpose

Terminates use of the GSL.

C Syntax

void gsterm_ ( )

FORTRAN Syntax

subroutine gsterm ( )

Pascal Syntax

PROCEDURE gsterm_ ( ) [PUBLIC];

Description

The **gsterm** subroutine terminates the GSL. It deallocates any private storage required, returns the virtual terminal to KSR Mode, and causes the monitor mode signals to be ignored.

gstext

Purpose

Writes annotated text.

C Syntax

int gstext_ (*x*, *y*, *number*, *text***)**

**int ****x*, ******y*, ******number*;
**char ****text*;

FORTRAN Syntax

INTEGER function gstext (*x*, *y*, *number*, *text***)**

INTEGER *x*, *y*, *number*
CHARACTER*n *text*

Pascal Syntax

FUNCTION gstext_ (

VAR *x*, *y*, *number*: **INTEGER;**
VAR *text*: **ARRAY** [1..*k*] **of CHAR**
): INTEGER [PUBLIC];

Description

The **gstext** subroutine writes the number of characters indicated by the parameters, starting at the specified baseline position and according to the relevant attributes. This subroutine is to be used only with annotated text.

The relevant attributes are:

- Color map
- Plane mask
- Font

- Code page
- Baseline direction
- Text color index.

Parameters

x, y	Define the baseline position for writing the text.
number	Indicates the number of bytes to write from the *text* string.
text	Contains the ASCII codes for the characters to write, as an array.

The graphics written to the frame buffer are determined by the 8-bit ASCII codes in the input data and the code page attribute. The ASCII control codes in between are ignored except the following: 1F, 1E, 1D, and 1C (hexadecimal). These control codes cause a shift to a predefined code page for the next ASCII character only. The code page definitions are:

1F	Bottom half of code page 1
1E	Top half of code page 1
1D	Bottom half of code page 2
1C	Top half of code page 2.

For any ASCII value between 0 and 31 (decimal), no graphic is written. For any other ASCII value and code page combination that does not result in a valid graphic, a dash is written.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length; the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_FBUF	Frame buffer overflow.
GS_INAC	Virtual terminal inactive.

gsulns

Purpose

Sets the user line pattern.

C Syntax

int gsulns_ (*pattern*, *length*, *begin***)**

**int ****pattern*, **length*, **begin*;

FORTRAN Syntax

INTEGER function gsulns (*pattern*, *length*, *begin***)**

INTEGER *pattern*, *length*, *begin*

Pascal Syntax

FUNCTION gsulns_ (

VAR *pattern*, *length*, *begin*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsulns** subroutine establishes the user line style.

Parameters

pattern — Defines the pixel pattern used for the line style. A 1 bit indicates that the GSL draws a pixel; a 0 bit means that it does not.

length — Defines the number of bits (starting with the most significant) of *pattern* used for line drawing. The bits are repeated for the length of the line.

The *length* parameter is a value not less than 2 or greater than 32.

begin — Indicates the length of the starting run of 1 bits in the pattern. It is used to adjust the beginning and ending runs of the non-continuous line styles.

For proper appearance, the application supplied line pattern should begin with a run of 1 bits and end with a run of 0 bits.

Return Value

GS_SUCC	Successful.
GS_LENG	Invalid length.

gsunlk

Purpose

Resumes signal processing.

C Syntax

void gsunlk_ ( )

FORTRAN Syntax

subroutine gsunlk ( )

Pascal Syntax

PROCEDURE gsunlk_ ( ) [PUBLIC];

Description

The **gsunlk** subroutine indicates to the GSL that the application is finished with the display adapter and it can now read the **SIGRETRACT** signal.

The application supplied routine called at **SIGRETRACT** can be entered as a result of **gsunlk**.

gsvgrn

Purpose

Sets the valuator granularity.

C Syntax

int gsvgrn_ (*valuators*, *granularity***)**

**char ****valuators*, ***granularity*;

FORTRAN Syntax

INTEGER function gsvgrn (*valuators*, *granularity***)**

CHARACTER *valuators*, *granularity*

Pascal Syntax

FUNCTION gsvgrn_ (

VAR *valuators*, *granularity*: **CHAR**
): INTEGER [PUBLIC];

Description

The **gsvgrn** subroutine sets the resolution of input events generated by the valuators, that is, the number of events per turn of the valuator dial.

Parameters

valuators Specifies which valuators to set to the indicated granularity. Each bit in *valuators* corresponds to one of the valuator dials, with the most significant bit indicating that valuator 0 is to be set, the next most significant bit indicating that valuator 1 is to be set, and so on.

granularity Specifies the desired resolution for the valuators indicated. It must have a value of 2 through 8 and indicates a resolution of 4, 8, 16, 32, 64, 128, or 256 points per revolution, respectively. The default value is 4, for a resolution of 16.

An attempt to set the valuator granularity may fail for a variety of reasons, the most likely of which is that the device is not attached. The problem can be determined with a specific **ioctl** to the virtual terminal. (See "hft" on page 6-23 for more information.)

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.
GS_VALG	Invalid granularity.

Related Information

In this book: "hft" on page 6-23

gsxblt

Purpose

Moves a rectangular block in system or display adapter memory from one location to another.

C Syntax

int gsxblt_ (*srcpix*, *dstpix*, *mskpix*, *W*, *H*, *logop***)**

int **srcpix*, **dstpix*, **mskpix*, **W*, **H*, **logop*;

FORTRAN Syntax

INTEGER function gsxblt (*srcpix*, *dstpix mskpix*, *W*, *H*, *logop***)**

INTEGER *srcpix*(*), *dstpix*(*), *mskpix*(*) *W*, *H*, *logop*

Pascal Syntax

FUNCTION gsxblt_ (

VAR *srcpix*, *dstpix*, *mskpix*: **ARRAY [32] of INTEGER;**
VAR *W*, *H*, *logop* **: INTEGER**
): INTEGER [PUBLIC];

Description

The **gsxblt** subroutine moves a rectangular block of pixels from one memory location to another, either in system memory or in the display adapter frame buffer.

For FORTRAN specific address information, see "gsxptr" on page 7-148.

The **gsxblt** subroutine is used to support windowing operations, such as overlays and movement around the screen. The source rectangle and the destination rectangle can be in either system or adapter pixel memory. The **gsxblt** subroutine is also used for user defined cursors and the save and restore of a pixel map for applications like pop-up menus.

The mask operation provided by the **gsxblt** subroutine controls which pixels in the destination rectangle can be modified.

The relevant attributes are:

- Plane mask
- Color map.

Parameters

srcpix Contains the address of the source pixel map.

dstpix Contains the address of the destination pixel map.

mskpix Contains the address of the mask operation pixel map. This parameter should equal zero if there is no bit mask operator to apply.

The *mskpix* pixel map must always consist of only one bit per pixel, and the mask rectangle must always be the same size as the source and destination rectangles. In the mask rectangle, a 1 bit means that the corresponding pixel in the destination rectangle can be modified, while a 0 bit means the destination pixel will not be modified.

W Defines the width of the rectangular area to be transferred.

H Defines the height of the rectangular area to be transferred.

logop Indicates the logical operation to perform between the source pixel map and the destination pixel map.

In the following table, please note:

- The source or tile (a special type of source) pixels represent bits of data to be merged in some way with the corresponding bits of data in the destination rectangle.
- The first three columns of the table specify the operations you can perform, and the **Code** column contains the corresponding value you should specify for the *logop* parameter.
- There are two unique codes for each logical operation, to be used depending on whether the tiling bit in the source pixel map is set. Codes 0-15 must be used when the tiling bit is not set, while codes 16-31 must be used when the tiling bit is set.
- A ~ (tilde) represents the logical INVERSE.

Type of Source	Logical Operation	Type of Destination	Code
		Destination clear	0
		Set Destination	15
	No operation	Destination	5
		~Destination	10
Source	REPLACE	Destination	3
Source	AND	Destination	1
Source	AND	~Destination	2
Source	EXCLUSIVE-OR	Destination	6
Source	OR	Destination	7
Source	OR	~Destination	11
~Source	REPLACE	Destination	12
~Source	AND	Destination	4
~Source	AND	~Destination	8
~Source	EXCLUSIVE-OR	Destination	9
~Source	OR	Destination	13
~Source	OR	~Destination	14
		Destination clear	16
		Set Destination	31
	No operation	Destination	21
		~Destination	26
Tile	REPLACE	Destination	19
Tile	AND	Destination	17
Tile	AND	~Destination	18
Tile	EXCLUSIVE-OR	Destination	22
Tile	OR	Destination	23
Tile	OR	~Destination	27
~Tile	REPLACE	Destination	28
~Tile	AND	Destination	20
~Tile	AND	~Destination	24
~Tile	EXCLUSIVE-OR	Destination	25
~Tile	OR	Destination	29
~Tile	OR	~Destination	30

A ***pixel map*** is a 32-bit array of integers that contains the following fields:

0 Device ID (0 for memory)

1 Flags

In the following explanations, bit 0 is the low-order bit.

- Plane (XY) format is selected when bit 0 is set and bits 1 and 2 are not set. Pixel (Z) format is selected when bits 0, 1, and 2 are not set.
- A repetitive tile is specified when bit 3 is set, while no tile is specified when bit 3 is not set.

  If the repetitive tile bit is set in the *srcpix*, pixel map, then the Device ID field in that pixel map must equal zero. The tile data must be in memory.
- Bit 4 selects the lower-left coordinate system when it is set, and the upper-left coordinate system when it is not set.

2 Height (in pixels)

3 Width (in pixels)

This value must be an even multiple of 16 pixels for all pixel maps, which means that all pixel maps must be at least 16 pixels wide.

4 Number of bits per pixel

5 Pixels per byte, right justified

6 Bytes per pixel

7 x offset

8 y offset

9 Address of upper-left corner of data

10 Foreground color index

11 Background color index

12 - 31 Reserved.

Definitions of pixel map terms include:

Device ID
This is a required parameter for all pixel map definitions. If the pixel map being defined is a display adapter, this field must contain the Device ID of that display adapter. If the pixel map resides in system memory, then this field must equal 0.

Pixel format
Data stored in this format has all bits for a pixel stored together. The data starts with the origin and increases first in the *x* direction, then in the *y* direction.

As an example using the upper-left coordinate system, a pixel map with four bits per pixel and one pixel per byte stores the four bits for the pixel at location (0,0) in the first byte of the data area, right justified in the byte. The four bits for the pixel at location (1,0) are stored in the second byte, followed by the rest of the pixel values in that row. When the end of the row is reached, the next byte contains the four bits for the pixel at location (0,1), followed by the rest of the pixel values in that row, and so on for the entire image.

Plane format

Plane format indicates that each of the bits that make up a pixel is stored in a separate, consecutive plane in memory. The most significant bit is first, followed by the next significant, and so on to the least significant bit, which is last. The bits within a plane are packed together 8 bits per byte. Therefore, using the upper-left coordinate system as an example, a pixel map with four bits per pixel would consist of four separate planes of data with the first bit value being the one for location (0,0) and increasing first in the *x* direction, then in the *y* direction.

Repetitive tiling operation

This operation consists of repeatedly copying a 16 pixel wide by 16 pixel high tile rectangle pointed to by the tile pixel map data address to fill a rectangular area of a size specified by the *H* and *W* parameters of this call. The format of the tile data is determined by the format defined in the *flags* field of the tile pixel map structure.

Upper-left coordinate system

This indicates that the upper-left corner of the pixel map is used as the origin of the coordinate system, with increasing values of *x* moving to the right and increasing values of *y* moving down. The *x offset* and *y offset* are to set the upper-left corner of the rectangle when using this coordinate system.

Lower-left coordinate system

This indicates that the lower-left corner of the pixel map is used as the origin of the coordinate system, with increasing values of *x* moving to the right and increasing values of *y* moving up. The *x offset* and the *y offset* are set to the lower-left corner of the rectangle when using this coordinate system. Note, however, that the *data address* specified in the pixel map structure must always point to the upper-left corner of the data area no matter which coordinate system is defined.

Number of bits per pixel

This field identifies the number of bits of data required to define a pixel value. For example, a simple monochrome display requires only one bit per pixel, while a color display may require four bits of information to define a pixel.

Number of pixels per byte

If the number of bits per pixel is less than 8, this field defines how many pixels are stored in each byte of pixel map data. A pixel map with only one bit per pixel must always store 8 pixels per byte. It is strongly recommended that for between 2 and 7 bits per pixel, you store data with only one pixel per byte.

Bytes per pixel
If the number of bits per pixel is greater than 8, this field defines how many bytes are used to store each pixel. It is strongly recommended that for between 9 and 16 bits per pixel, you store data two bytes per pixel. For between 17 and 32 bits per pixel, data should be stored four bytes per pixel.

Foreground color index
This specifies the color index value to use for a value of 1 in the source pixel map during a color expansion operation.

Background color index
This specifies the color index value to use for a value of 0 in the source pixel map during a color expansion operation.

A ***color expansion operation*** takes place automatically when the source pixel map data area contains only one bit per pixel and the destination pixel map data area is a color display adapter frame buffer defined to have more than one bit per pixel. In this case, when a 1 is specified in the source pixel map data area, the *foreground color index* value specified in the destination pixel map (*dstpix*) is written to the destination data area. When a 0 is specified in the source pixel map data area, the *background color index* value specified in the destination pixel map (*dstpix*) is written to the destination data area.

The *foreground color index* and the *background color index* must be initialized in the *dstpix* pixel map before calling this operation, but do not need to be initialized in the *srcpix* or *mskpix* pixel maps.

Not all logical operations are supported for a color expansion operation. The following table shows which operations are supported. In this table, a ~ (tilde) represents the logical INVERSE. Note that the operations in the left column of the table are for source pixel maps, while the operations in the right column are for tile pixel maps.

Type of Operation	Code	Type of Operation	Code
Destination clear	0	Destination clear	16
Set destination	15	Set destination	31
Destination	5	Destination	21
~Destination	10	~Destination	26
Source	3	Tile	19
~Source	12	~Tile	28

If a source or destination pixel map structure defines the active display adapter, you do not need to initialize all the fields of that pixel map structure. Device-dependent information, such as *height*, *width*, *pixels per byte*, *bytes per pixel*, and *address of data*, is supplied automatically. You must initialize the fields for *device ID*, *bits per pixel*, *flags* (except for the data format bits), *x offset*, and *y offset*. Also, the *foreground color index* and the *background color index* must be initialized if appropriate for this adapter.

When initializing a pixel map structure to use as the *mskpix* parameter:

1. The *flags* field should equal a value of 0x01 if the upper left coordinate system will be used or 0x11 if the lower left coordinate system will be used.
2. The number of *bits per pixel* must equal 1.
3. The number of *pixels per byte* must equal 8.

The GSL plane mask attribute applies to all **gsxblt** operations that use the display adapter as the source or destination pixel map.

The GSL color map attribute applies to all **gsxblt** operations that use the display adapter as the destination pixel map.

Return Value

GS_SUCC	Successful.
GS_IWID	Invalid width specification. The *x_offset* plus the *W* parameter of one of the pixel maps exceeds the total width of that pixel map.
GS_IHEI	Invalid height specification. The *y_offset* plus the *H* parameter of one of the pixel maps exceeds the total height of that pixel map.
GS_NPLF	Source and destination data formats do not match.
GS_INAC	Virtual terminal inactive.
GS_CORD	Invalid coordinate specified that placed the origin of the source, destination, or mask rectangle outside its pixel map.
GS_IBPP	Invalid value for *bits per pixel* in the source pixel map.
GS_CEXP	Color expansion operation attempted, but the destination pixel map was not a display adapter.
GS_PWID	The width of one of the pixel maps is not an even multiple of 16 pixels.

Related Information

In this book: “gsxptr” on page 7-148.

gsxcnv

Purpose

Converts pixel map data organization.

C Syntax

int gsxcnv_ (*inppix*, *outpix***)**

**int ****inppix*, ***outpix*;

FORTRAN Syntax

INTEGER function gsxcnv (*inppix*, *outpix***)**

INTEGER *inppix***(*)**, *outpix***(*)**

Pascal Syntax

FUNCTION gsxcnv_ (

VAR *inppix*, *outpix*: **INTEGER**
): INTEGER [PUBLIC];

Description

The **gsxcnv** subroutine converts pixel map data to and from planes. That is, **gsxcnv** converts XY form to and from pixels, or Z form.

See "gsxblt" on page 7-139 for more information on data formats and the definition of a pixel map.

Parameters

inppix	Points to the address of the pixel map whose data area is to be converted.
outpix	Points to the address of the pixel map that contains the address of where to put the converted data.

Both the *inppix* and *outpix* parameters contain the address of a pixel map. The fields of each pixel map must be completely initialized before calling this subroutine. Both pixel maps must point to data areas that reside in system memory, not in a display adapter frame buffer.

The *inppix* and *outpix* pixel maps do not have to specify the same number of bits per pixel. If there are more input bits per pixel, the least significant bits are truncated. If there are less input bits per pixel than required to fill out the destination, the most significant bits are filled with zeros.

The **gsxcnv** subroutine only supports pixel maps defined to have 8 bits per pixel or less. If a pixel format pixel map is defined with less than 8 bits per pixel, the data must be arranged 1 byte per pixel, right justified in that byte.

The widths and heights of the two data areas must be identical.

Warning: The calling process must allocate enough storage in the area pointed to by the *outpix* pixel map to contain all of the converted data.

Return Value

GS_SUCC	Successful.
GS_INPF	Invalid data format specified in *inppix* pixel map structure.
GS_OUTF	Invalid data format specified in *outpix* pixel map structure.
GS_BMAX	Pixel map defines data of more than 8 bits per pixel.

Related Information

In this book: “gsxblt” on page 7-139.

gsxptr

Purpose

Handles FORTRAN addressing of data.

C Syntax

None

FORTRAN Syntax

INTEGER function gsxptr (*intptr*, *datptr*)

INTEGER *intptr*(*), *datptr*(*)

Pascal Syntax

None

Description

The **gsxptr** subroutine places a data address in a variable so that the data address field of a pixel map structure can be initialized.

In a FORTRAN application, you must first call the **gsxptr** subroutine, then the **gsxblt** subroutine.

For C and Pascal applications and for more information, see "gsxblt" on page 7-139.

Parameters

intptr	Contains the address of the variable containing the data area.
datptr	Will be initialized to the address of the data area.

Return Value

GS_SUCC	Successful.

Related Information

In this book: "gsxblt" on page 7-139.

Chapter 8. Sockets

About This Chapter

This chapter contains information about the sockets function provided by AIX.

The information in this chapter is divided into two sections: "Overview" and "Socket Routines." The "Overview" section provides introductory material, definitions, and sources of additional information, while the "Socket Routines" section contains reference information on the subroutines that perform socket operations.

If you are not familiar with the use of sockets as a communication tool, you should read the entire "Overview" section first, then refer to the reference material as needed. If you are familiar with sockets, you may want to scan the information in the overview to see how the facility works in AIX.

Overview

A ***socket*** is an object that provides communications between processes. Sockets are referenced by file descriptors and have qualities similar to those of a character special device. Read, write, and select operations can be performed on sockets by using the appropriate system calls.

A socket is created with the **socket** subroutine. (See "socket" on page 8-47.) This subroutine creates a socket of a specified domain, type, and protocol. Sockets have different qualities depending on these specifications.

A ***domain*** is a name space or an address space. Each domain has different rules for valid names and interpretation of names. After a socket is created, it can be given a name, according to the rules of the domain in which it was created.

AIX provides support for the following socket domains:

Local Provides socket communication between processes running on the same AIX system when a domain of **AF_UNIX** is specified. A name in this domain is a string of ASCII characters whose maximum length is machine dependent.

Internet Provides socket communication between a local process and a process running on a remote host when a domain of **AF_INET** is specified. This domain requires that the IBM RT PC Interface Program for use with TCP/IP be installed on your system. A name in this domain is a DARPA Internet address, made up of a 32-bit IP address and a 16-bit port address. (See the discussion of addresses and names in *Interface Program for use with TCP/IP.*)

In AIX, there are two types of sockets:

SOCK_DGRAM Provides ***datagrams***, which are connectionless messages of a fixed maximum length. This type of socket is generally used for short messages, such as a name server or time server, since the order and reliability of message delivery is not guaranteed.

In the local domain, **SOCK_DGRAM** is similar to a message queue. In the Internet domain, **SOCK_DGRAM** is implemented on the UDP/IP protocol.

SOCK_STREAM Provides sequenced, two-way byte streams with a transmission mechanism for out-of-band data. The data is transmitted on a reliable basis, in order.

In the local domain, **SOCK_STREAM** is like a pipe. In the Internet domain, **SOCK_STREAM** is implemented on the TCP/IP protocol.

A ***protocol*** is used only is more than one protocol is supported in this domain. Otherwise, this parameter is set to 0.

Socket Names

A socket name, which is also called a socket address, is specified by the **sockaddr** structure. This structure is defined in the **sys/socket.h** header file, and it contains the following members:

```
ushort  sa_family;    /* Defines socket address family */
char    sa_data[14];  /* Contains up to 14 bytes of direct address */
```

The **sa_family** is the address family or domain, either **AF_UNIX** for the local domain or **AF_INET** for the Internet domain. The contents of **sa_data** depend on the protocol in use, but generally a socket name consists of a machine name part and a port or service name part.

Related Network Publications

For general information about networking, the following publications are recommended. These publications are distributed by the Network Information Center on behalf of the Defense Communications Agency and Defense Advanced Research Projects Agency (DARPA). The mailing address is:

Network Information Center
SRI International
Menlo Park, CA 92025

- *Assigned Numbers*, RFC990, J. Reynolds, J. Postel
- *Broadcasting Internet Datagrams*, RFC919, J. Mogul
- *Domain Names - Concepts and Facilities*, RFC882, P. Mockapetris
- *Domain Names - Implementation and Specification*, RFC883, P. Mockapetris
- *File Transfer Protocol*, RFC959, J. Postel
- *Internet Control Message Protocol*, RFC792, J. Postel
- *Internet Name Server Protocol*, IEN116, J. Postel
- *Internet Protocol*, RFC791, J. Postel
- *Internet Standard Subnetting Procedure*, RFC950, J. Mogul
- *Name/Finger*, RFC742, K. Harrenstien
- *Official ARPA-Internet Protocols*, RFC944, J. Reynolds, J. Postel
- *Simple Mail Transfer Protocol*, RFC821, J. Postel
- *Telnet Binary Transmission*, RFC856, J. Postel, J. Reynolds
- *Telnet Option Specifications*, RFC855, J. Postel, J. Reynolds
- *Telnet Protocol Specification*, RFC854, J. Postel, J. Reynolds
- *Telnet Terminal Type Option*, RFC930, M. Solomon, E. Wimmers
- *The TFTP Protocol*, RFC783, K. R. Sollins
- *Time Protocol*, RFC868, J. Postel, K. Harrenstien
- *Transmission Control Protocol*, RFC793, J. Postel
- *Trivial File Transfer Protocol*, RFC783, K. R. Sollins
- *User Datagram Protocol*, RFC768, J. Postel

Socket Routines

The following section contains reference material for each of the subroutines that perform socket operations. These subroutines are listed in alphabetical order according to routine name.

accept

Purpose

Accepts a connection on a socket.

Library

Sockets Library (**libsock.a**)

Syntax

#include <sys/types.h>
#include <sys/socket.h>

int accept (*s*, *addr*, *addrlen***)**
int *s*;
struct sockaddr **addr*;
int **addrlen*;

Description

The **accept** subroutine extracts the first connection on the queue of pending connections, creates a new socket with the same properties as *s*, and allocates a new file descriptor for that socket. If no pending connections are present on the queue and the calling socket is not marked as non-blocking, **accept** blocks the caller until a connection is present. If the socket specified by *s* is marked non-blocking and there are no connections pending on the queue, **accept** returns an error as described below. The accepted socket cannot be used to accept more connections. The original socket, *s*, remains open and can accept more connections.

The *s* parameter is a socket that was created with the **socket** subroutine, was bound to an address with the **bind** subroutine, and has issued a successful call to the **listen** subroutine.

The *addr* parameter is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of *addr* is determined by the domain in which the communication occurs. The *addrlen* parameter initially contains the amount of space pointed to by the *addr* parameter. On return, it contains the actual length (in bytes) of the address returned. This subroutine is used with connection-based socket types, such as **SOCK_STREAM**.

Before calling the **accept** subroutine, you can find out if the socket is ready to accept the connection by doing a read select with the **select** system call.

Return Value

Upon successful completion, the non-negative socket descriptor of the accepted socket is returned. If the **accept** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF The *s* parameter is not valid.

ENOTSOCK The *s* parameter refers to a file, not a socket.

EOPNOTSUPP The referenced socket is not of type **SOCK_STREAM**.

EFAULT The *addr* parameter is not in a writable part of the user address space.

EWOULDBLOCK The socket is marked non-blocking, and no connections are present to be accepted.

Related Information

In this book: "select" on page 2-111, "bind" on page 8-9, "connect" on page 8-11, "listen" on page 8-36, and "socket" on page 8-47.

bind

Purpose

Binds a name to a socket.

Library

Sockets Library (**libsock.a**)

Syntax

#include <sys/types.h>
#include <sys/socket.h>

int bind (*s*, *name*, *namelen***)**
int *s*;
**struct sockaddr ****name*;
int *namelen*;

Description

The **bind** subroutine assigns a name to an unnamed socket. When a socket is created with the **socket** subroutine, it belongs to the address family specified in the **socket** call, but has no name assigned yet. The **bind** subroutine requests that *name* be assigned to the socket.

Note that all named sockets must have unique names. A socket does not have to have a name before it can make a connection to another socket, and a socket returned by the **accept** subroutine already has a name assigned to it by the subroutine.

Return Value

Upon successful completion, a value of 0 is returned. If the **bind** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF	The *s* parameter is not valid.
ENOTSOCK	The *s* parameter refers to a file, not a socket.
EADDRNOTAVAIL	The specified address is not available from the local machine.
EADDRINUSE	The specified address is already in use.
EINVAL	The socket is already bound to an address.
EACCESS	The requested address is protected, and the current user does not have permission to access it.
EFAULT	The *addr* parameter is not in a writable part of the user address space.

Related Information

In this book: "connect" on page 8-11, "listen" on page 8-36, "socket" on page 8-47, and "getsockname" on page 8-28.

connect

Purpose

Initiates a connection on a socket.

Library

Sockets Library (**libsock.a**)

Syntax

#include <sys/types.h>
#include <sys/socket.h>

int connect (*s*, *name*, *namelen*)
int *s*;
**struct sockaddr ****name*;
int *namelen*;

Description

The *s* parameter is a socket. If it is of type **SOCK_DGRAM**, then this subroutine permanently specifies the peer, the socket to which datagrams are sent. This allows use of the **send** subroutine rather than **sendto** or **sendmsg**, which require the address that the socket data should be sent to.

If the socket is of type **SOCK_STREAM**, which provides read and write capability between sockets, this subroutine attempts to make a connection to another socket. The other socket is specified by *name*, which is an address in the communications space of the socket. Each communications space interprets the *name* parameter in its own way.

Return Value

Upon successful completion, a value of 0 is returned. If the **connect** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF The *s* parameter is not valid.

ENOTSOCK The *s* parameter refers to a file, not a socket.

EADDRNOTAVAIL The specified address is not available from the local machine.

EAFNOSUPPORT The addresses in the specified address family cannot be used with this socket.

EISCONN The socket is already connected.

ETIMEDOUT The establishment of a connection timed out before a connection was made.

ECONNREFUSED The attempt to connect was rejected.

ENETUNREACH The network is not reachable from this host.

EADDRINUSE The specified address is already in use.

EFAULT The *addr* parameter is not in a writable part of the user address space.

EWOULDBLOCK The socket is marked non-blocking, and the connection cannot be immediately completed. You can select the socket for writing during the connection process.

Related Information

In this book: "select" on page 2-111, "accept" on page 8-7, "socket" on page 8-47, and "getsockname" on page 8-28.

gethostbyaddr, gethostbyname, sethostent, endhostent

Purpose

Gets network host entry.

Library

Sockets Library (**libsock.a**)

Syntax

```
#include <netdb.h>

struct hostent *gethostbyaddr (addr, len, type)
char *addr;
int len, type;

struct hostent *gethostbyname (name)
char *name;

void sethostent (stayopen)
int stayopen;

void endhostent ( )
```

Description

The **gethostent**, **gethostbyname**, and **gethostbyaddr** subroutines each return a pointer to an object. This object is a **hostent** structure, which contains information obtained from a name server program, or a field from a line in the **/etc/hosts** file (the network host data base).

The **gethostbyname** subroutine recognizes either domain name servers (as described in RFC883) or IEN116 name servers. (The details of these name servers are contained in publications listed in "Related Network Publications" on page 8-5.) If the file **/etc/resolv.conf** exists, a domain name server is assumed by **gethostbyname**. (See "resolv.conf" on page 5-68.3.) If **/etc/resolv.conf** does not exist, an IEN116 name server is assumed.

When domain name servers are used and the server request times out, the local **/etc/hosts** file is checked. When an IEN116 name server is used, the **/etc/hosts** file is checked *before* the server is queried. Note that the **gethostbyaddr** subroutine can only use a domain name server.

The **hostent** structure is defined in the **netdb.h** header file, and it contains the following members:

```
        char    *h_name;         /* official name of host   */
        char    **h_aliases;     /* alias list              */
        int     h_addrtype;      /* host address type       */
        int     h_length;        /* length of address       */
        char    **h_addr_list;   /* list of addresses       */

    #define h_addr  h_addr_list[0]   /* address, for backward compatibility */
```

The members of the structure are defined below:

h_name Official name of the host.

h_aliases An array, terminated with a 0, of alternate names for the host.

h_addrtype The type of address being returned. The subroutine always sets this value to **AF_INET**.

h_length The length of the address in bytes.

h_addr_list An array, terminated by a 0, of pointers to the network addresses for the host. Host addresses are returned in network byte order.

h_addr The first address in h_addr_list, provided for backward compatibility.

The **sethostent** subroutine opens and rewinds the file. If the *stayopen* parameter is 0, the host data base is closed after each call to the **gethostbyname** or **gethostbyaddr** subroutines. Otherwise, the file is not closed after each call.

The **endhostent** subroutine closes the file.

The **gethostbyname** and **gethostbyaddr** subroutines query the nameserver or search the file sequentially from its beginning until finding a matching host name or host address, or until encountering the end of the file. Host addresses are supplied in network order.

Return Value

The **gethostbyname** and **gethostbyaddr** subroutines return a pointer to a **hostent** structure on success.

Note: The return value points to static data that is overwritten by subsequent calls.

A **NULL** pointer (**0**) is returned if an error occurs or the end of the file is reached and the **h_errno** variable is set to indicate the error.

Diagnostics

The **gethostbyname** and **gethostbyaddr** subroutines fail if one or more of the following are true:

HOST_NOT_FOUND The host specified by the *name* parameter was not found.

TRY_AGAIN The local server did not receive a response from an authoritative server. Try again later.

NO_RECOVERY This error code indicates an unrecoverable error.

NO_ADDRESS The requested *name* is valid but does not have an Internet address at the name server.

File

/etc/hosts Host name data base.
/etc/resolv.conf Name server and domain name data base.

Related Information

In this book: "hosts" on page 5-58.2, "resolv.conf" on page 5-68.3, and "Related Network Publications" on page 8-5.

gethostid, sethostid

Purpose

Gets or sets the unique identifier of the current host.

Library

Sockets Library (**libsock.a**)

Syntax

```
int gethostid ( )
```

```
int sethostid (hostid)
int hostid;
```

Description

The **gethostid** subroutine returns the 32-bit identifier for the current host, as set by **sethostid**.

The **sethostid** subroutine establishes a 32-bit identifier for the current host that is intended to be unique. Often, this is a DARPA Internet address for the local machine.

This subroutine can only be used by processes with an effective user ID of superuser. In the AIX Operating System, the host ID is usually set by the **netconfig** portion of the Interface Program for use with TCP/IP, using the **/etc/net** configuration file.

Return Value

Upon successful completion, the **gethostid** subroutine returns the identifier for the current host, and the **sethostid** subroutine returns a value of 0. If the **gethostid** or **sethostid** subroutine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The **gethostid** or **sethostid** subroutine fails if the following is true:

EINVAL	There are no IP interfaces available. IBM RT PC Interface Program for use with TCP/IP is not installed on this system.

The **sethostid** subroutine also fails if the following is true:

EPERM	The calling process did not have an effective user ID of superuser.

Related Information

In this book: "getsockname" on page 8-28.

The **hostname** command in *Interface Program for use with TCP/IP.*

gethostname, sethostname

Purpose

Gets or sets the name of the current host.

Library

Sockets Library (**libsock.a**)

Syntax

int gethostname (*name*, *namelen***)**
**char ****name*;
int *namelen*;

int sethostname (*name*, *namelen***)**
**char ****name*;
int *namelen*;

Description

The **gethostname** subroutine returns the standard host name of the current host, as set by **sethostname**. The parameter *namelen* specifies the size of the *name* array. The returned name is null-terminated unless insufficient space is provided.

The **sethostname** subroutine sets the name of the host machine *name* with the length *namelen*. This subroutine can only be used by processes with an effective user ID of superuser. In the AIX Operating System, the host name of a machine is usually set by the Interface Program for use with TCP/IP in its initialization program (**/etc/rc.tcpip**).

Return Value

Upon successful completion, a value of 0 is returned. If the **gethostname** or **sethostname** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EFAULT	The *name* parameter or *namelen* parameter gives an address that is not valid.
EPERM	The calling process did not have an effective user ID of superuser.

Related Information

In this book: "gethostid, sethostid" on page 8-16.

The **hostname** command in *Interface Program for use with TCP/IP*.

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent

Purpose

Gets network entry.

Library

Sockets Library (**libsock.a**)

Syntax

#include <netdb.h>

struct netent *getnetent ( )

struct netent *getnetbyname (*name***)**
char ****name*;**

struct netent *getnetbyaddr (*net***)**
long *net***;**

void setnetent (*stayopen***)**
int *stayopen***;**

void endnetent ( )

Description

The **getnetent**, **getnetbyname**, and **getnetbyaddr** subroutines each return a pointer to an object. This object is a **netent** structure, which contains the field of a line in the **/etc/networks** file (the network data base). The **netent** structure is defined in the **netdb.h** header file, and it contains the following members:

```
char    *n_name;       /* official name of net */
char    **n_aliases;   /* alias list           */
int     n_addrtype;    /* net number type      */
long    n_net;         /* net number           */
```

The members of the structure are defined below:

n_name Official name of the network.

n_aliases An array, terminate with a zero, of alternate names for the network.

n_addrtype The type of network number being returned. This value must be **AF_INET**.

n_net The network number. Network numbers are returned in machine byte order.

The **getnetent** subroutine reads the next line of the file. If the file is not open, **getnetent** opens it.

The **setnetent** subroutine opens and rewinds the file. If the *stayopen* parameter is 0, the net data base is closed after each call to **getnetent**. Otherwise, the file is not closed after each call.

The **endnetent** subroutine closes the file.

The **getnetbyname** and **getnetbyaddr** subroutines search the file sequentially from its beginning until finding a matching net name or net number, or until encountering the end of the file. Network numbers are supplied in host order.

Return Value

A pointer to a **netent** structure is returned on success.

Note: The return value points to static data that is overwritten by subsequent calls.

A **NULL** pointer (**0**) is returned if an error occurs or the end of the file is reached.

File

/etc/networks Network name data base.

Related Information

In this book: "networks" on page 5-66.2.

getpeername

Purpose

Gets the name of the connected peer.

Library

Sockets Library (**libsock.a**)

Syntax

int getpeername (*s*, *name*, *namelen***)**
int *s*;
**struct sockaddr ****name*;
**int ****namelen*;

Description

The **getpeername** subroutine returns the name of the ***peer***, or connected socket, that is connected to the socket specified by the *s* parameter. You should initialize the *namelen* to indicate the amount of space pointed to by *name*. On return, it contains the actual size of the name returned (in bytes).

Return Value

Upon successful completion, a value of 0 is returned. If the **getpeername** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF	The *s* parameter is not valid.
ENOTSOCK	The *s* parameter refers to a file, not a socket.
ENOTCONN	The socket is not connected.
ENOBUFS	Insufficient resources were available in the system to complete the call.

EFAULT The *addr* parameter is not in a writable part of the user address space.

Related Information

In this book: "bind" on page 8-9, "socket" on page 8-47, and "getsockname" on page 8-28.

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent

Purpose

Gets protocol entry.

Library

Sockets Library (**libsock.a**)

Syntax

#include <netdb.h>

struct protoent *getprotoent ( )

struct protoent *getprotobyname (*name*)
**char ****name*;

struct protoent *getprotobynumber (*proto*)
int *proto*;

void setprotoent (*stayopen*)
int *stayopen*;

void endprotoent ( )

Description

The **getprotoent**, **getprotobyname**, and **getprotobynumber** subroutines each return a pointer to an object. This object is a **protoent** structure, which contains the field of a line in the **/etc/protocols** file (the network protocol data base). The **protoent** structure is defined in the **netdb.h** header file, and it contains the following members:

```
char    *p_name;        /* official name of protocol */
char    **p_aliases;    /* alias list                */
long    p_proto;        /* protocol number           */
```

The members of the structure are defined below:

p_name Official name of the protocol.

p_aliases An array, terminated by a 0, of alternate names for the protocol.

p_proto The protocol number.

The **getprotoent** subroutine reads the next line of the file. If the file is not open, **getprotoent** opens it.

The **setprotoent** subroutine opens and rewinds the file. If the *stayopen* parameter is 0, the protocol data base is not closed after each call to **getprotoent**. Otherwise, the file is not closed after each call.

The **endprotoent** subroutine closes the file.

The **getprotobyname** and **getprotobynumber** subroutines search the file sequentially from its beginning until finding a matching protocol name or protocol number, or until encountering the end of the file.

Return Value

A pointer to a **protoent** structure is returned on success.

Note: The return value points to static data that is overwritten by subsequent calls.

A **NULL** pointer (**0**) is returned if an error occurs or the end of the file is reached.

File

/etc/protocols	Protocol name data base.

Related Information

In this book: "protocols" on page 5-68.1.

getservent, getservbyname, getservbyport, setservent, endservent

Purpose

Gets service entry.

Library

Sockets Library (**libsock.a**)

Syntax

#include <netdb.h>

struct servent *getservent ( )

struct servent *getservbyname (*name*, *proto***)**
**char ****name*, ******proto*;

struct servent *getservbyport (*port*, *proto***)**
int *port*;
**char ****proto*;

void setservent (*stayopen***)**
int *stayopen*;

void endservent ( )

Description

The **getservent**, **getservbyname**, and **getservbyport** subroutines each return a pointer to an object. This object is a **servent** structure, which contains the field of a line in the **/etc/services** file (the network services data base). The **servent** structure is defined in the **netdb.h** header file, and it contains the following members:

```
char   *s_name;      /* official name of service   */
char   **s_aliases;  /* alias list                 */
long   s_port;       /* port where service resides */
char   *s_proto;     /* protocol to use            */
```

The members of the structure are defined below:

s_name Official name of the service.

s_aliases An array, terminated by a 0, of alternate names for the service.

s_port The port number at which the service resides. Port numbers are returned in network byte order.

s_proto The name of the protocol to use when contacting the service.

The **getservent** subroutine reads the next line of the file. If the file is not open, **getservent** opens it.

The **setservent** subroutine opens and rewinds the file. If the *stayopen* parameter is 0, the service data base is closed after each call to **getservent**. Otherwise, the file is not closed after each call.

The **endservent** subroutine closes the file.

The **getservbyname** and **getservbyport** subroutines search the file sequentially from its beginning until finding a matching protocol name or port number, or until encountering the end of the file. When a protocol name is also supplied, searches also match the protocol.

Return Value

A pointer to a **servent** structure is returned on success.

Note: The return value points to static data that is overwritten by subsequent calls.

A **NULL** pointer (**0**) is returned if an error occurs or the end of the file is reached.

File

/etc/services Service name data base.

Related Information

In this book: "getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent" on page 8-24 and "services" on page 5-68.5.

getsockname

Purpose

Gets the socket name.

Library

Sockets Library (**libsock.a**)

Syntax

int getsockname (*s*, *name*, *namelen***)**
int *s*;
struct sockaddr **name*;
int **namelen*;

Description

The **getsockname** subroutine stores the current name for the socket specified by the *s* parameter into the structure pointed to by the *name* parameter. Initialize the value pointed to by the *namelen* parameter to indicate the amount of space pointed to by *name*. On return, the *namelen* parameter points to the actual size of the name returned (in bytes).

Return Value

Upon successful completion, a value of 0 is returned. If the **getsockname** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF The *s* parameter is not valid.

ENOTSOCK The *s* parameter refers to a file, not a socket.

ENOBUFS Insufficient resources were available in the system to complete the call.

EFAULT The *addr* parameter is not in a writable part of the user address space.

Related Information

In this book: "bind" on page 8-9 and "socket" on page 8-47.

getsockopt, setsockopt

Purpose

Gets and sets options on sockets.

Library

Sockets Library (**libsock.a**)

Syntax

#include <sys/types.h>
#include <sys/socket.h>

int getsockopt (*s*, *level*, *optname*, *optval*, *optlen***)**
int *s*, *level*, *optname*;
char **optval*;
int **optlen*;

int setsockopt (*s*, *level*, *optname*, *optval*, *optlen***)**
int *s*, *level*, *optname*;
char **optval*;
int **optlen*;

Description

The **getsockopt** and **setsockopt** subroutines manipulate options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

When manipulating socket options, you must specify the level at which the option resides and the name of the option. To manipulate options at the socket level, specify *level* as SOL_SOCKET. To manipulate options at any other level, supply the appropriate protocol number for the protocol controlling the option. For example, to indicate that an option will be interpreted by the TCP protocol, set *level* to the protocol number of TCP, as defined in the **sys/socket.h** header file.

Use the parameters *optval* and *optlen* to access option values for *setsockopt.* For **setsockopt**, these parameters identify a buffer in which the value for the requested option or options are returned. For **getsockopt**, the *optlen* parameter initially contains the size of the buffer pointed to by the *optval* parameter. On return, it is modified to indicate the actual size of the value returned. If no option value is supplied or returned, the *optval* parameter can be zero.

The *optname* parameter and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The **sys/socket.h** header file contains definitions for socket level options. These options are:

SO_DEBUG	Turns on recording of debugging information.
SO_ACCEPTCONN	Specifies that socket is listening.
SO_REUSEADDR	Allows local address reuse.
SO_KEEPALIVE	Keeps connections active.
SO_DONTROUTE	Does not apply routing on outgoing messages.
SO_LINGER	Lingers on a **close** system call if data is present.
SO_OOBINLINE	Leaves received ***out-of-band data*** (data marked urgent) in line.
SO_SNDBUF	Sends buffer size.
SO_RCVBUF	Receives buffer size.
SO_SNDLOWAT	Sends low-water mark.
SO_RCVLOWAT	Receives low-water mark.
SO_SNDTIMEO	Sends timeout.
SO_RCVTIMEO	Receives timeout.
SO_ERROR	Gets error status.
SO_TYPE	Gets socket type.

SO_DEBUG enables debugging in the underlying protocol modules. **SO_REUSEADDR** indicates that the rules used in validating addresses supplied by a **bind** subroutine should allow reuse of local addresses. **SO_KEEPALIVE** enables the periodic transmission of messages on a connected socket. If the connected socket fails to respond to these messages, the connection is broken and processes using that socket are notified with a **SIGPIPE** signal. **SO_DONTROUTE** indicates that outgoing messages should bypass the standard routing facilities and are directed to the appropriate network interface according to the network portion of the destination address. **SO_LINGER** controls the action taken when unsent messages are queued on a socket and a **close** system call is performed. If **SO_LINGER** is set, the system blocks the process during the **close** system call until it can transmit the data or until the time expires. Specify the amount of time for the linger interval by using the **setsockopt** subroutine when requesting **SO_LINGER**. If **SO_LINGER** is not specified and a **close** system call is issued, the system handles the call in a way that allows the process to continue as quickly as possible.

Options at other protocol levels vary in format and name.

Return Value

Upon successful completion, a value of 0 is returned. If the **getsockopt** or **setsockopt** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF The *s* parameter is not valid.

ENOTSOCK The *s* parameter refers to a file, not a socket.

ENOPROTOOPT The option is unknown.

EFAULT The *addr* parameter is not in a writable part of the user address space.

Related Information

In this book: "socket" on page 8-47 and "getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent" on page 8-24.

htonl, htons, ntohl, ntohs

Purpose

Convert values between host and Internet network byte order.

Library

Sockets Library (**libsock.a**)

Syntax

```
#include <sys/types.h>
#include <netinet/in.h>

unsigned long htonl (hostlong)
unsigned long hostlong;

unsigned short htons (hostshort)
unsigned short hostshort;

unsigned long ntohl (netlong)
unsigned long netlong;

unsigned short ntohs (netshort)
unsigned short netshort;
```

Description

These subroutines convert 16- and 32-bit quantities between network byte order and host byte order.

These subroutines are often used in conjunction with Internet addresses and ports as returned by the **gethostent** and **getservent** subroutines.

Related Information

In this book: "gethostbyaddr, gethostbyname, sethostent, endhostent" on page 8-13 and "getservent, getservbyname, getservbyport, setservent, endservent" on page 8-26.

inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof

Purpose

Manipulation subroutines for Internet addresses.

Library

Sockets Library (**libsock.a**)

Syntax

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

struct in_addr inet_addr (cp)
char *cp;

int inet_network (cp)
char *cp;

char *inet_ntoa (in)
struct in_addr in;

struct in_addr inet_makeaddr (net, lna)
int net, lna;

int inet_lnaof (in)
struct in_addr in;

int inet_netof (in)
struct in_addr in;
```

Description

The **inet_addr** and **inet_network** subroutines each interpret character strings representing numbers expressed in the Internet standard dot (.) notation, returning numbers suitable for use as Internet addresses and Internet network numbers. The *cp* parameter represents a string of characters in the Internet address form.

The **inet_ntoa** subroutine takes an Internet address and returns an ASCII string representing the address in dot notation. The *in* parameter contains the Internet address to be converted to ASCII.

The **inet_makeaddr** takes an Internet network number and a local network address and constructs an Internet address from it. The *net* parameter contains an Internet network number, while the *lna* parameter contains a local network address.

The **inet_netof** and **inet_lnaof** subroutines break apart Internet addresses, returning the network number and local network address part. The *in* parameter represents the Internet address to separate.

All Internet addresses are returned in network order, with the first byte being the high-order byte. All network numbers and local addresses are returned as integer values in machine format.

The values specified using the dot notation take one of the following forms:

```
a.b.c.d
a.b.c
a.b
a
```

When four parts are specified, each is interpreted as a byte of data and assigned to the four bytes of an Internet address, ordered from high-order to low-order.

When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as **128.***net.host.*

When a two-part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as *net.host.*

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied for each part of a dot notation may be decimal, octal, or hexadecimal, as specified in C language. A leading `0x` or `0X` implies hexadecimal, a leading `0` implies octal, and anything else is interpreted as decimal.

Return Value

The **inet_addr** and **inet_network** subroutines return numbers suitable for use as Internet addresses and Internet network numbers, respectively, on success. If the **inet_addr** or **inet_network** subroutine fails, a value of -1 is returned.

Related Information

In this book: "hosts" on page 5-58.2, "networks" on page 5-66.2, "gethostbyaddr, gethostbyname, sethostent, endhostent" on page 8-13, and "getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent" on page 8-20.

listen

Purpose

Listens for connections on a socket.

Library

Sockets Library (**libsock.a**)

Syntax

int listen (*s*, *backlog***)**
int *s*, *backlog***;**

Description

To accept connections, create a socket with **socket**, specify a backlog for incoming connections with **listen**, and accept the connections with **accept**. The **listen** subroutine applies only to sockets of type **SOCK_STREAM**.

The *backlog* parameter defines the maximum length for the queue of pending connections. If a connection request arrives with the queue full, the client receives an error with an indication of **ECONNREFUSED**.

Return Value

Upon successful completion, a value of 0 is returned. If the **listen** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF	The *s* parameter is not valid.
ENOTSOCK	The *s* parameter refers to a file, not a socket.
EOPNOTSUPP	The referenced socket is not of type that supports **listen**.

Related Information

In this book: "accept" on page 8-7, "connect" on page 8-11, and "socket" on page 8-47.

recv, recvfrom, recvmsg

Purpose

Receives a message from a socket.

Library

Sockets Library (**libsock.a**)

Syntax

```
#include <sys/types.h>
#include <sys/socket.h>

int recv (s, buf, len, flags)
int s;
char *buf;
int len, flags;

int recvmsg (s, msg, flags)
int s;
struct msghdr msg[ ];
int flags;

int recvfrom (s, buf, len, flags, from, fromlen)
int s;
char *buf;
int len, flags;
struct sockaddr *from;
int *fromlen;
```

Description

The **recv** subroutine can be used only on a connected socket (see "connect" on page 8-11), but **recvfrom** and **recvmsg** can be used to receive data on a socket whether it is connected or not.

If the value of *from* is anything other than zero, the source address of the message is filled in. The *fromlen* parameter is initialized to the size of the buffer associated with the *from* parameter. On return, it is modified to indicate the actual size of the address stored there. These subroutines return the length of the message. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from. For more information, see "socket" on page 8-47.

If no messages are available at the socket, the receive subroutines wait for a message to arrive, unless the socket is non-blocking. If a socket is non-blocking, a -1 is returned with the external variable **errno** set to **EWOULDBLOCK**.

Use the **select** system call to determine when more data arrives. For more information, see "select" on page 2-111.

The *flags* argument to receive a call is formed by logically OR-ing one or more of the values shown in the following list:

MSG_PEEK Peeks at incoming message.

MSG_OOB Processes out-of-band data.

The **recvmsg** subroutine uses a **msghdr** structure to minimize the number of directly supplied parameters. The **msghdr** structure is defined in the **sys/socket.h** header file, and it contains the following members:

```
caddr_t msg_name;         /* optional address            */
int     msg_namelen;      /* size of address             */
struct  iov *msg_iov;     /* scatter/gather array        */
int     msg_iovlen;       /* # of elements in msg_iov    */
caddr_t msg_accrights;    /* access rights sent/received */
int     msg_accrightslen; /* length of access rights     */
```

In the above structure, the fields are defined as follows:

msg_name Defines the destination address if the socket is unconnected. If no names are needed, you can use a **NULL** pointer for **msg_name**.

msg_namelen Specifies the size of **msg_name**.

msg_iov Describes the scatter gather locations.

msg_iovlen Specifies the number of elements in the **msg_iov** array.

msg_accrights Defines the access rights sent with the message.

msg_accrightslen Specifies the length of the access rights.

Return Value

Upon successful completion, the length of the message in bytes is returned. If the **recv**, **recvfrom**, or **recvmsg** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF	The *s* parameter is not valid.
ENOTSOCK	The *s* parameter refers to a file, not a socket.
EWOULDBLOCK	The socket is marked non-blocking, and no connections are present to be accepted.
EINTR	The receive was interrupted by delivery of a signal before any data was available for the receive.
EFAULT	The *addr* parameter is not in a writable part of the user address space.

Related Information

In this book: "send, sendto, sendmsg" on page 8-43 and "socket" on page 8-47.

rexec

Purpose

Allows command execution on a remote host.

Library

Sockets Library (**libsock.a**)

Syntax

int rexec (*host*, *port*, *user*, *passwd*, *command*, *errfdp***)**

char ******host*;
int *port*;
char **user*, **passwd*, **command*;
int **errfdp*

Description

The **rexec** subroutine allows the calling process to execute commands on a remote host.

The *host* parameter contains the name of a remote host that is listed in the **/etc/hosts** file. If the name of the host is not found in this file, the **rexec** fails.

The *port* parameter specifies the well-known DARPA Internet port to use for the connection. A pointer to the structure that contains the necessary port can be obtained by issuing the following call:

```
getservbyname("exec", "tcp")
```

The protocol for the connection is described in detail in the discussion of **rexecd** in *Interface Program for use with TCP/IP.*

The *user* and *passwd* parameters point to a user ID and password valid at the host. If these parameters are not supplied, the **rexec** subroutine takes the following actions until finding a user ID and password to send to the remote host:

1. Searches the current environment for the user ID and password on the remote host.
2. Searches the user's home directory for a file called **.netrc** that contains a user ID and password.
3. Prompts the user for a user ID and password.

The *command* parameter points to the name of the command to be executed at the remote host.

If the connection succeeds, a socket in the Internet domain of type **SOCK_STREAM** is returned to the calling process and is given to the remote command as standard input and standard output.

If *errfdp* is not 0, an auxiliary channel to a control process is set up, and a descriptor for it is placed in **errfdp*. The control process provides diagnostic output from the remote command on this channel and also accepts bytes as signal numbers to be forwarded to the process group of the command. This diagnostic information does not include remote authorization failure, since this connection is set up after authorization has been verified.

If *errfdp* is 0, then the standard error of the remote command is the same as standard output, and no provision is made for sending arbitrary signals to the remote process. In this case, however, it may be possible to send out-of-band data to the remote command.

Return Value

The **rexec** subroutine fails and a value of -1 is returned if the specified host name does not exist.

Related Information

In this book: "hosts" on page 5-58.2.

The discussion of **rexecd** in *Interface Program for use with TCP/IP.*

send, sendto, sendmsg

Purpose

Sends a message from a socket.

Library

Sockets Library (**libsock.a**)

Syntax

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int send (s, msg, len, flags)
int s;
char *msg;
int len, flags;
```

```
int sendmsg (s, msg, flags)
int s;
struct msghdr msg[ ];
int flags;
```

```
int sendto (s, msg, len, flags, to, tolen)
int s;
char *msg;
int len, flags;
struct sockaddr *to;
int tolen;
```

Description

The **send** subroutine sends a message only when the socket is in a connected state. The **sendto** and **sendmsg** subroutines can be used at any time.

Give the address of the target by *to*, with *tolen* specifying its size. Specify the length of the message with *len*. If the message is too long to pass through the underlying protocol, the error **EMSGSIZE** is returned and the message is not transmitted.

No indication of failure to deliver is implied in a **send**. Return values of -1 indicate some locally detected errors.

If no space for messages is available at the sending socket to hold the message to be transmitted, the **send** subroutine blocks unless the socket is in a non-blocking I/O mode.

Use the **select** system call to determine when it is possible to send more data.

The *flags* argument to send a call is formed by logically OR-ing one or both of the values shown in the following list:

MSG_OOB	Processes out-of-band data on sockets that support **SOCK_STREAM**.
MSG_DONTROUTE	Sends without using routing tables.

For a description of the **msghdr** structure, see "recv, recvfrom, recvmsg" on page 8-38.

Return Value

Upon successful completion, the number of characters sent is returned. If the **send**, **sendto**, or **sendmsg** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF	The *s* parameter is not valid.
ENOTSOCK	The *s* parameter refers to a file, not a socket.
EFAULT	The *addr* parameter is not in a writable part of the user address space.
EMSGSIZE	The socket requires that the message be sent all at once, and the message is too large for that to happen.
EWOULDBLOCK	The socket is marked non-blocking, and no connections are present to be accepted.

Related Information

In this book: "recv, recvfrom, recvmsg" on page 8-38 and "socket" on page 8-47.

shutdown

Purpose

Shuts down part or all of a full-duplex connection.

Library

Sockets Library (**libsock.a**)

Syntax

int shutdown (*s*, *how***)**
int *s*, *how*;

Description

The **shutdown** subroutine allows you to disable receives, sends, or both on the socket specified by the *s* parameter. The action of the subroutine is determined by the *how* parameter, according to the following values:

0 Disallows further receives.

1 Disallows further sends.

2 Disallows both further sends and receives.

Return Value

Upon successful completion, a value of 0 is returned. If the **shutdown** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EBADF	The *s* parameter is not valid.
ENOTSOCK	The *s* parameter refers to a file, not a socket.
ENOTCONN	The socket is not connected.

Related Information

In this book: "connect" on page 8-11 and "socket" on page 8-47.

socket

Purpose

Creates an endpoint for communication and returns a descriptor.

Library

Sockets Library (**libsock.a**)

Syntax

#include <sys/types.h>
#include <sys/socket.h>

int socket (*af*, *type*, *protocol*)
int *af*, *type*, *protocol*;

Description

The **socket** subroutine creates an endpoint for communication and returns a socket descriptor.

The *af* parameter specifies an address format with which addresses specified in later socket operations should be interpreted. These formats are defined in the **sys/socket.h** header file. The formats are:

AF_UNIX	AIX path names
AF_INET	ARPA Internet addresses.

The value of the *type* parameter specifies the semantics of communication. AIX supports these types:

SOCK_STREAM Provides sequenced, two-way byte streams with a transmission mechanism for out-of-band data.

SOCK_DGRAM Provides ***datagrams***, which are connectionless messages of a fixed maximum length (usually small).

The *protocol* parameter specifies a particular protocol to be used with the socket. In most cases, a single protocol exists to support a particular socket type using a given address format. When many protocols exist, you must specify a particular protocol. Use the number for the communication domain in which the communication takes place.

The different types of sockets available are used for different purposes. **SOCK_DGRAM** sockets allow sending datagrams to correspondents named in **send** socket calls. Programs can also receive datagrams via sockets by using the **recv** subroutines.

SOCK_STREAM sockets are full-duplex byte streams. A stream socket must be connected before any data may be sent or received on it. Create a connection to another socket with the **connect** routine. Once connected, use the **read** and **write** system calls, or the **send** and **recv** subroutines to transfer data. Issue the **close** system call when a session is finished. Use the **send** and **recv** subroutines for out-of-band data.

SOCK_STREAM communications protocols are designed to prevent the loss or duplication of data. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable period of time, the connection is broken. When this occurs, the socket routines indicate an error with a return value of -1 and with **ETIMEDOUT** as the specific code written to the global variable **errno**. If a process sends on a broken stream, a **SIGPIPE** signal is raised. Processes that cannot handle the signal terminate.

When out-of-band data arrives on a socket, a **SIGURG** signal is sent to the process group. The process group associated with a socket may be read or set by the **SIOCGPGRP ioctl** operation or the **SIOCSPGRP ioctl** operation. If you want to receive a signal on any data, use both the **SIOCSPGRP** and **FIOASYNC ioctl** operations. These **ioctl** operations are defined in the **bsd/sys/ioctl.h** file.

Sockets can be set to either ***blocking*** or ***non-blocking*** I/O mode. The **FIONBIO ioctl** operation is used to determine this mode. When **FIONBIO** is set, the socket is marked non-blocking. If a read is tried and the desired data is not available, the socket does not wait for the data to become available, but returns immediately with the error code **EWOULDBLOCK**. When **FIONBIO** is not set, the socket is in blocking mode. In this mode, if a read is tried and the desired data is not available, the calling process waits for the data.

Similarly, when writing, if **FIONBIO** is set and the output queue is full, an attempt to write causes the process to return immediately with an error code of **EWOULDBLOCK**.

The operation of sockets is controlled by socket level options. The **getsockopt** and **setsockopt** subroutines are used to get and set these options, which are defined in the **sys/socket.h** file. See “getsockopt, setsockopt” on page 8-30 for information on how to use these options.

Return Value

Upon successful completion, a descriptor referring to the socket is returned. If the **socket** routine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EAFNOSUPPORT The addresses in the specified address family cannot be used with this socket.

ESOCKNOSUPPORT The socket in the specified address family is not supported.

EMFILE The **per-process** descriptor table is full.

ENOBUFS Insufficient resources were available in the system to complete the call.

Related Information

In this book: "ioctl" on page 2-56, "select" on page 2-111, "accept" on page 8-7, "bind" on page 8-9, "connect" on page 8-11, "getsockname" on page 8-28, "getsockopt, setsockopt" on page 8-30, "listen" on page 8-36, "recv, recvfrom, recvmsg" on page 8-38, "send, sendto, sendmsg" on page 8-43, "shutdown" on page 8-45, and "socketpair" on page 8-50.

socketpair

Purpose

Creates a pair of connected sockets.

Syntax

```
#include <sys/types.h>
#include <sys/socket.h>

socketpair (d, type, protocol, sv)
int d, type, protocol;
int sv[2];
```

Description

The **socketpair** subroutine creates an unnamed pair of connected sockets in the specified domain *d*, of the specified *type*, and using the optionally specified *protocol*. The descriptors used in referencing the new sockets are returned in *sv*[0] and *sv*[1]. The two sockets are identical.

Note: The **socketpair** subroutine can be used only in the local (**AF_UNIX**) domain. This subroutine does not create sockets for use in the Internet domain.

Return Value

Upon successful completion, a value of 0 is returned. If the **socketpair** subroutine fails, a value of -1 is returned, and **errno** is set to indicate the error.

Diagnostics

The subroutine fails if one or more of the following are true:

EMFILE	This process has too many descriptors in use.
EAFNOSUPPORT	The addresses in the specified address family cannot be used with this socket.
EPROTONOSUPPORT	The specified protocol cannot be used on this system.
EOPNOSUPPORT	The specified protocol does not allow create of socket pairs.

EFAULT	The *sv* parameter is not in a writable part of the user address space.

Appendix A. Error Codes

This section describes the error conditions that can occur when using the system calls described in this book. Some subroutines that invoke system calls indicate errors in a similar way.

System calls indicate the fact that an error has occurred by returning a special value. This value is almost always -1, but check the description of the individual system call to be sure. Also, a number that identifies the error is stored in an external variable named **errno**. The **errno** variable is not cleared when a system call finishes successfully, so its value is meaningful only after an error has occurred.

If you are going to check the value of **errno** in a program, include the following line at the top of the source file:

```
#include <errno.h>
```

The **errno.h** header file declares the **errno** variable and defines the name of each error condition.

For each error code, the following list shows the symbolic name defined in the **/usr/include/errno.h** header file, the corresponding numeric value, and a brief description of the error:

EPERM (1) **Not the owner**

Cause: You attempted to modify a file in some way forbidden except to the owner of the file or to superuser. Or, a user other than superuser attempted to do something that only superuser is allowed to do.

ENOENT (2) **No such file or directory**

Cause: The file specified does not exist, or one of the directories in a path name does not exist.

ESRCH (3) **No such process**

Cause: A process, corresponding to that specified in the *pid* parameter of the **kill** or **ptrace** system calls, cannot be found.

EINTR (4) **Interrupted system call**

Cause: An asynchronous signal (such as interrupt or quit), which you have elected to catch, occurred during a system call. If the system call resumes after processing the signal, it appears as if the interrupted system call returned this error condition.

EIO (5) **I/O error**

Cause: A physical I/O error occurred. In some cases, this error occurs on a system call following the one to which it actually applies.

ENXIO (6) **No such device or address**

Cause: I/O on a special file referred to a device or subdevice that does not exist or referred to an address that is beyond the limits of the device.

E2BIG (7) **Argument list too long**

Cause: The combined length of the argument list and the environment list passed to one of the **exec** system calls totaled more than 5,120 bytes.

ENOEXEC (8) **Exec format error**

Cause: A request was made to execute a file that has the appropriate permissions, but does not start with either a valid shared library magic number, or a valid text header. (For information about text headers, see "a.out" on page 4-5.)

EBADF (9) **Bad file number**

Cause: A file descriptor was specified that does not refer to an open file, or a read request was made to a file that is open only for writing, or a write request was made to a file that is open only for reading.

ECHILD (10) **No child processes**

Cause: A process that invoked the **wait** system call has no existing child processes that have not been waited for.

EAGAIN (11) **No more processes**

Cause: The **fork** system call failed because the system's process table is full or the user is not allowed to create any more processes. Or, an attempt was made to access a region of a file that has an outstanding enforcement-mode lock. (See "lockf" on page 2-64 about file locking.)

ENOMEM (12) **Not enough space**

Cause: During a **brk**, **sbrk**, or **exec** system call, a program asked for more space than the system is able to supply. This is not a temporary condition. The maximum space size is a system parameter.

EACCES (13) **Permission denied**

Cause: An attempt was made to access a file in a way that is forbidden by the protection system.

EFAULT (14) **Bad address**

Cause: An address passed to a system call that points to a location outside of the process's allocated address space.

ENOTBLK (15) **Block device required**

Cause: A nonblock file was specified when a block device is required, such as in the **mount** system call.

EBUSY (16) **Mount device busy**

Cause: An attempt was made to mount a device that is already mounted, or an attempt was made to dismount a device on which there is an active file. This error also occurs when an attempt is made to enable accounting when it has already been enabled.

EEXIST (17) **File exists**

Cause: An existing file was specified to a system call or subroutine that would create that file, such as the **link** system call.

EXDEV (18) **Cross-device link**

Cause: An attempt was made to link to a file on another device. (See "link" on page 2-62.)

ENODEV (19) **No such device**

Cause: An attempt was made to use an inappropriate system call to a device, for example, to write to a read-only device.

ENOTDIR (20) **Not a directory**

Cause: A nondirectory parameter was specified where a directory is required, for example in a path prefix or as a parameter to the **chdir** system call.

EISDIR (21) **Is a directory**

Cause: An attempt was made to write on a directory.

EINVAL (22) **Invalid parameter**

Cause: An invalid parameter or action was specified to a system call, such as dismounting a device that is not mounted, specifying an undefined signal, or writing to a file for which **lseek** has generated a negative file pointer.

ENFILE (23) **File table overflow**

Cause: An attempt was made to open a file, and the system's table of open files is full.

EMFILE (24) **Too many open files**

Cause: A process attempted to open more than two hundred (200) file descriptors at one time.

ENOTTY (25) **Not a typewriter**

Cause: An **ioctl** system call was issued to a special file that does not support **ioctl**.

ETXTBSY (26) **Text file busy**

Cause: This error occurs when an attempt is made to execute a pure-procedure program or shared library that is currently open for writing or reading. It also occurs when an attempt is made to open a pure-procedure program or shared library for writing while that program or library is being executed.

EFBIG (27) **File too large**

Cause: The size of a file exceeded the maximum file size (1,082,201,088 bytes), or the maximum size set by the **ulimit** system call.

ENOSPC (28) **No space left on the device**

Cause: During a write to an ordinary file, the device ran out of free space.

ESPIPE (29) **Illegal seek**

Cause: An **lseek** system call was issued to an "unseekable" file or device, such as a pipe.

EROFS (30) **Read-only file system**

Cause: An attempt was made to modify a file or directory on a device that is mounted as read-only.

EMLINK (31) **Too many links**

Cause: An attempt was made to make more than the maximum number of links (1000) to a file.

EPIPE (32) **Broken pipe**

Cause: An attempt was made to write to a pipe for which there is not a process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

EDOM (33) **Math argument**

Cause: A parameter to a Math Library (**libm.a**) subroutine was out of the domain of the function.

ERANGE (34) **Result too large**

Cause: The return value of a Math Library (**libm.a**) subroutine is not representable within machine precision.

ENOMSG (35) **No message of the desired type**

Cause: An attempt was made to receive a message of a type that does not exist on the specified message queue.

EIDRM (36) **Identifier removed**

Cause: The specified identifier has been removed from the file system's name space. (See "msgctl" on page 2-73, "semctl" on page 2-115, and "shmctl" on page 2-135.)

Note: The values **ECHRNG (37)** through **EL2HLT (44)** are supplied in the **errno.h** header file for compatibility with UNIX System V. These values are not set by any AIX software.

ECHRNG (37)	**Channel number out of range**
EL2NSYNC (38)	**Level 2 not synchronized**
EL3HLT (39)	**Level 3 halted**
EL3RST (40)	**Level 3 reset**
ELNRNG (41)	**Link number out of range**
EUNATCH (42)	**Protocol driver not attached**
ENOCSI (43)	**No CSI structure available**
EL2HLT (44)	**Level 2 halted**

EDEADLK (45) **Potential deadlock**

Cause: A potential deadlock was detected while attempting to lock a region of a file with the **lockf** system call.

ENOTREADY(46) **Device not ready**

Cause: The device is not ready for operation. For example, a diskette drive does not contain a diskette, or the device is not powered on.

EWRPROTECT(47) **Write-protected media**

Cause: The I/O media is write-protected.

EFORMAT(48) **Unformatted or incompatible media**

Cause: The I/O media has not been formatted or the format is not compatible with the I/O device.

ENOLCK(49) **No locks available**

Cause: There are not more file locks available. Too many segments are already locked.

ENOCONNECT(50) **New connection not made**

Cause: A new network connection to a remote node cannot be made.

EBADCONNECT(51) **Connection not found**

Cause: An attempt to use an existing connection to a remote node has failed.

ESTALE(52) **No file system**

Cause: The file system of a remote file has been unmounted, or the file descriptor of a remote file has become obsolete.

EDIST(53) **Requests blocked**

Cause: Sending or receiving of requests is currently not allowed. All requests may be blocked or only requests of the specified type.

EWOULDBLOCK (54) **Resource not available**

Cause: The socket is not blocking because **O_NDELAY** is set, but the desired kind of data is not available or, for an **accept** operation, no connections are pending.

EINPROGRESS (55) **Connection in progress**

Cause: The socket was marked **O_NDELAY** by an **fcntl** system call, then a **connect** operation was attempted that has not completed yet.

EALREADY (56) **Already in progress**

Cause: The requested socket connection or disconnection is already in progress.

ENOTSOCK (57) **Not a socket**

Cause: The command cannot complete because the file descriptor specified is not a socket.

EDESTADDRREQ (58) **Destination address required**

Cause: The attempted socket operation failed because a destination address was required, but not provided.

EMSGSIZE (59) **Message too long**

Cause: The socket data transfer failed because the message exceeded the size limits.

EPROTOTYPE (60) **Incorrect protocol type**

Cause: Either the two sockets to be connected are not of the same type, or the protocol used does not support this type of socket.

ENOPROTOOPT (61) **Unavailable protocol option**

Cause: The protocol specified either does not support this particular option or does not support any options.

EPROTONOSUPPORT (62) **Protocol not available**

Cause: No protocol of the specified type and domain exists.

ESOCKTNOSUPPORT (63) **Socket not supported**

Cause: The type of socket specified is not supported. Do not use this type of socket in your program.

EOPNOTSUPP (64) **Operation not supported**

Cause: This socket, with its particular type, domain, and protocol, does not allow the requested operation.

EPFNOSUPPORT (65) **Protocol not supported**

Cause: The socket protocol specified is not supported. Do not use this protocol in your program.

EAFNOSUPPORT (66) **Invalid socket name**

Cause: The socket name is of a type that is not valid in this socket or the domain.

EADDRINUSE (67) **Socket name in use**

Cause: A **bind** or **connect** operation was attempted using a socket name that is already in use.

EADDRNOTAVAIL (68) **Socket name not available**

Cause: The requested socket name is not available to this machine. Either an incorrect socket name was used, or there is a problem at the remote node where the socket name should be.

ENETDOWN (69) **Network down**

Cause: A socket operation failed because the network is down.

ENETUNREACH (70) **Remote node unreachable**

Cause: A socket operation failed because the destination is at a remote node that cannot be reached over the network.

ENETRESET (71) **Remote node reset**

Cause: The host the socket was connected to went down. The connection can be re-established after the remote node is restarted.

ECONNABORTED (72) **Connection terminated**

Cause: The connection between a socket and a remote node was terminated at the local node, the remote node, or the network level.

ECONNRESET (73) **Connection reset**

Cause: The connection with another socket was reset by that socket. This **errno** can be set due to an error, or just due to a connection that was closed.

ENOBUFS (74) **Insufficent buffer space**

Cause: Not enough buffer space is available for the requested socket operation.

EISCONN (75) **Socket already connected**

Cause: A **connect** operation was attempted on a socket that is already connected.

ENOTCONN (76) **Socket not connected**

Cause: A socket operation other than a **connect** was attempted on a socket that is not currently connected, or a **send** operation that does not require a connection was attempted without a destination address.

ESHUTDOWN (77) **Socket already shut down**

Cause: An attempt was made to send data after a **shutdown** operation was done on the socket.

ETIMEDOUT (78) **Connection timed out**

Cause: A remote socket did not respond within the timeout period set by the protocol of the socket on this node.

ECONNREFUSED (79) **Connection refused**

Cause: A remote node refused to allow the attempted **connect** operation.

EHOSTDOWN (80) **Host down**

Cause: A socket operation failed because the remote node specified is down.

EHOSTUNREACH (81) **Host unreachable**

Cause: A socket operation failed because no route to the remote node was available due to an incorrect address, an incorrect routing table, or network hardware problems.

Appendix B. Writing a Queuing System Backend

This section assumes that you know what a queue **backend** is, **friendly** and **unfriendly** types, and that you need to write one. It discusses friendly backends, not backends in general. Backend in this chapter refers to **friendly** backends. See *Managing the AIX Operating System* for more information about backends.

Introduction

The principal purpose of a backend is to send characters to a device, typically a printer. There are several ways the backend can do this. First, it can open a particular device and write to it. This has the advantage of simplicity, but it means that the backend cannot be used for any other device. Second, it can accept a parameter supplied by the user to tell it which device to use. This is more flexible, but involves a little extra work. Third, it can simply write to its standard output, and the **qdaemon** command will automatically open the device onto the correct file descriptor. This is the recommended method. It works only if the **file** field in the **qconfig** file has been set up appropriately.

The backend is invoked once for every file or group of files to be printed. The name of each file to be printed is passed to the backend as a parameter. The backend must open the file, read its contents, and send them to the device in one of the ways previously described.

Since the backend must open files, read them, and write to devices, you (the writer of a backend) should understand the domain where the backend operates. When a backend is invoked, its current directory is the one where the print request was made. The name of the file or files to be printed can either be a direct or relative path name. The UID and GID of the backend are those of the process that invoked the **print** command.

If the backend writes to its standard output and allows the **qdaemon** process to open the device, permissions are handled by the **qdaemon** automatically. Otherwise, the backend will need to have write permission on the special file corresponding to the device. This may require changing the protections on the device or installing the backend set-user-ID or set-group-ID.

By default, **stdin, stdout,** and **stderr** are all open to the null device (**/dev/null**), though it is possible to override the setting of **stdout** (and possibly **stdin**) with the **file** and **access** lines in the **qconfig** file.

Interaction Between Qdaemon and Backend

Besides reading files and writing to devices, a friendly backend must cooperate with the **qdaemon** in several ways. The requirements can be summarized as follows:

- Recognize a **-statusfile** parameter and call a library routine that does some initialization.
- Print burst pages as requested.
- Print extra copies as requested.
- Update status information (pages printed, percentage done) periodically.
- Supply charges (accounting data) for the completed job.
- Exit with some agreed on codes.
- Pass error messages through a special routine.
- Set state to WAITING, if appropriate.
- Terminate cleanly on receipt of SIGTERM.

Each requirement is discussed more fully in the following.

There is a set of library routines that the backend should use to fulfill these requirements. The routines were designed to make the task of writing a backend as easy as possible. These routines are in the **/lib/libqb.a** library and accessible with the **-lqb** flag. The individual routines are discussed in the body of the text that follows, and a summary table is given at the end of this chapter.

The -statusfile Parameter

When the **qdaemon** process invokes a backend, it passes the following parameters, in order:

1. The parameters appearing in the **qconfig** file
2. The **-statusfile** parameter, if running as a friendly backend
3. The flags that the **print** command did not recognize, in the order they were given
4. The names of one or more files to be printed.

The presence of the **-statusfile** parameter indicates that the status file is open on file descriptor 3 of the backend.

The status file provides a means for the **qdaemon** process and the backend to communicate. The daemon passes such information as the date of the file, which burst pages are to be printed, the number of copies to be printed, and so on. The backend passes back the charge for the job it has just finished running. In addition, the backend periodically writes into the file the number of pages it has printed and what percent of the job is finished. This information is read by the **print -q** command.

Backends should never explicitly write into their status file. Instead, they should call the library routines that do so. The reason for calling the routines is twofold: (1) backends are spared the trouble of accessing the status file directly, and (2) the format of the status file can be changed without requiring backends to rewritten. In this case, the backends only need to be re-linked.

To initialize certain data common to the library routines, the backend must call the routine **log_init**. The call is:

```
log_init();
```

This routine should be called when the **-statusfile** parameter is recognized. The **log_init** routine, like all the routines in library whose names begin **log_**, returns a value of -1 if it fails.

Burst Pages

There are four types of burst pages:

header A page preceding a file that shows its title, date, it recipient, and other information.

trailer A page following a file that gives the name of the user of the output.

feed Blank pages printed only when the printer has become idle. Feed pages make it easier for users to tear off paper from the printer.

align A form-feed printed only when the printer has been idle and is about to print a new job. The form-feed aligns the paper to top-of-form and is helpful if someone has moved the paper while the printer was idle.

If the backend will never print any burst pages, the following information can be skipped.

The printing of burst pages is done automatically by the **burst_page** routine. The routine takes two parameters: the address of a function and the width of the header and/or trailer desired. If the function address is NULL (#include <stdio.h>), the routine uses the supplied function and passes the character as its single parameter.

By passing the address of a special function for output, a backend can maintain strict control of what goes to the device and when it goes to the device. For example, the **burst_pages** routine uses line-feeds to separate lines, and form-feeds to separate pages. If the device requires a carriage return to precede every line-feed, the special function can make such a translation.

The basic algorithm for synchronizing calls to the **burst_page** routine with file printing looks like this:

```
burst_page(fnaddr, width);
while (files are to be printed)
{
   burst_page(fnaddr, width);
   print the next file;
   burst_page(fnaddr, width);
}
```

Every backend should follow this structure. The line numbers are used for reference in the following explanation.

The **burst_page** routine uses the information in the status file to decide whether (and how) to print a header, a trailer, some feed pages, or an aligning form-feed. The status file is set up by the **qdaemon**, using the information provide in the **qconfig** file. For example, if the **qconfig** file contains the line *header=group*, the call on line (2) results in a header page only if this file is used by a different user than the user who printed the previous file on this device. The **burst_page** routine when invoked on line (2) makes that test and either prints the header or returns. Similarly, line (3) either prints a trailer or does nothing.

With the exception of line (1), which may appear to extraneous, the algorithm is simple. This first call is necessary because **qdaemon** does not ask the backend to print a **group** trailer until it knows positively that there are no more files for a particular user. It cannot know this fact until either the first file for the next user is ready to be printed or there are no more files for this device. In the first case, **qdaemon** appends the trailer request for the previous user to the file request for the current one. Line (1) prints the trailer for the previous user if the **trailer=group** option has been selected; otherwise, it does nothing. In the second case, the backend is invoked with no file parameter at all. In this case, line (1) prints both a trailer and feed pages (assuming **qconfig** requests them), the **while** test fails, and the backend exits.

The **burst_page** routine assumes that the printer is at the top of the page, and it prints a form-feed at the end of its header or trailer to leave the printer in the same state. Backends are responsible for maintaining the position of the paper. The **align** option is useful only for device like continuous-form daisy-wheel printers, where it is possible for the printer paper to be out of alignment after a job is removed.

The **burst_page** routine should be enough for most friendly backends. If it is not, the library provides a set of routine at a lower level that should prove helpful for generating burst pages. There is a group of routines that return information from the status file, and two other routines that print headers and trailers, respectively.

Functions in the first group take no parameters; the following describes their actions:

get_align
Returns **TRUE** or **FALSE**, telling whether an alignment form-feed is to be printed, assuming **get_newuser()** is **TRUE** and **get_endgroup()** is **FALSE**.

get_endgroup
Returns **TRUE** or **FALSE**, telling whether this the end of a group of files for the same user.

get_feed Returns the number of feed pages to be printed, assuming **get_endgroup()** is **TRUE** and **get_newuser()** is **FALSE**.

get_from
Returns the name of the person that made the print request.

get_header
Returns **NEVER**, **ALWAYS**, or **GROUP** (**#include <IN/backend.h>**).

get_lastuser
Returns the name of the previous user, assuming **get_endgroup()** is **TRUE**.

get_moddate
Returns a string showing the modification date of the file.

get_newuser
Returns **TRUE** or **FALSE**, telling whether this is the beginning of a group of files for a new user.

get_nodeid
Returns the node ID.

get_qdate
Returns a string showing the date that the request was queued.

get_title Returns the title of the job being printed.

get_to Returns the name of the person for whom the job is intended.

get_trailer
Returns **NEVER**, **ALWAYS**, or **GROUP**.

In addition, there is a routine **put_header**(*fnaddr, width*), that prints a header with no following form-feed, returning the number of lines printed, and a routine, **put_trailer**(*user,fnaddr,width*), that prints a trailer for **user**, again with no following form-feed, and returns the number of lines printed. The **fnaddr** and **width** parameters work like the same parameters in the **burst_page** function previously stated.

It should be emphasized that the auxiliary functions should not be necessary for most backends. The **burst_page()** routine handles all tasks required when it is called as described in the previous algorithm.

Extra Copies

The user can request that extra copies of a file be printed with the **print** *-nc* command. The *print -nc = 5 filename* command prints 5 copies of a file.

The **print** program passes the *-nc* information to the **qdaemon** process, which puts it into the status file. Backends should get the information by calling the **get_copies()** routine, which returns the total number of copies desired.

Job Status Information

The **print**-*q* command displays information about currently running jobs, including its origniator, its title, the number of pages to be printed, and the percentage completed. All this information comes from the status file. Most of the information is set up by the **qdaemon** process when the backend is first invoked, except the **pages printed** and **percent done** fields, which must be filled in by the backend itself.

To provide this information, the backend should periodically call **log_progress(***pages, percent***)**, which writes the two numbers in the appropriate place in the file. The backend is free to call this routine as frequently or infrequently as desired; once at the end of each page is recommended.

Charge for the Job

Whenever a backend completes a job, the **qdaemon** process reads the status file for a charge. If the **qconfig** file has been set up appropriately, the charge is written to a file that is eventually processed by the accounting programs, resulting in a bill (real or imaginary) for the user issuing the print request.

The backend passes the charge back to the **qdaemon** process with the routine **log_charge(***charge***)**, where *charge* is a long integer. The backend should certainly call this routine on exit. It should also call the routine along with **log_progress** while printing the job. Otherwise, if the job is canceled, no charge will be made for the pages printed up to that point.

The charge is interpreted by all current accounting programs as the number of pages printed. However, a backend might decide that one page on its device is worth two or three normal pages (or some fraction) and set the charge accordingly.

Exit Codes

When a backend exists, the **qdaemon** process looks at its exit code for information about whether the job was completed successfully, whether the device is still usable, and so on. Therefore, it is important that backends use the same convention for their exit codes. The backend should use **#include <IN/standard.h>** for the values of the codes mentioned here.

The permissible exit codes are:

EXITOK No problems were encountered.

EXITBAD
The parameters were bad in some way. That is, a flag was unrecognizable or illegal, a file could not be opened, and so on. The **qdaemon** process notifies the user, throws out the job request, and continues sending jobs to the device.

EXITERROR
The backend could not finish printing the job and that it wants another chance. The **qdaemon** process restarts the same job (from the beginning) on the same device. The **qdaemon** process enforces a limit on the number of times that the job will be restarted.

EXITFATAL
The job could not be finished because of a problem in the device that requires manual intervention. The **qdaemon** process sets the state of the device (displayed by **print -q**) to OFF, sends a message to the console, and does not run any further jobs on that device until someone has explicitly set its state to ON again (with a **print -du**).

EXITSIGNAL
The backend was interrupted by a **SIGTERM** signal.. (**#include <signal.h>**).

Return Error Messages

If the backend cannot run a job (that is if it exits EXITBAD or EXITFATAL), it should send a message to the **qdaemon** process explaining the problem. The **qdaemon** process passes the message to the user, and, for EXITFATAL prints it on the the console.

The message should be sent with the **log_message** routine, which takes parameters in the style of **printf**:

```
log_message("cannot open file %s; error return %d\n",
                filename, erret);
```

The message cannot be longer than MAXMESG (**#include <IN/backend.h>**) bytes.

Set State to WAITING

The **print** *-q* command displays the status of a particular device. One of the entries in the table that is displayed shows whether the device is READY, RUNNING, WAITING, or OFF. This information is taken from the status file.

Normally, the **qdaemon** process keeps the status file updated, and a backend need never worry about it. However, some backends may want to explicitly set the state to WAITING (**#include <IN/backend.h>**) if they can no longer send output to the device, and set it back to RUNNING when output resumes. For example, a backend that paused at the end of each page, waiting for the user to load the next page and type a RETURN, might want to set the status to WAITING during this time.

The **log_status**(*status*) routine can be used to change the status of the job from RUNNING to WAITING and back again. The parameter is the new status.

Terminate on Receipt of SIGTERM

When a user cancels a running job with **print**-*ca*, the **print** command passes the request to the **qdaemon** process. Therefore, in order for cancellation to work, the backend must terminate soon after receipt of the signal. There are two ways to comply with this requirement.

First, the backend cannot do anything special about SIGTERM, in which case the signal kills the backend process immediately. This option is the simplest, but does not allow the backend to do any cleanup (reset line speeds, put paper at top-of-form, hang up the phone) before it terminates.

Second, the backend can catch SIGTERM, carry out whatever cleanup tasks are required, and exit EXITSIGNAL (**#include <IN/standard.h>**). The special exit code tells the **qdaemon** process that the job was canceled.

Backends that decide to catch SIGTERM should exit very soon after receipt of the signal. If the cleanup code is too long, or if it can hang indefinitely (waiting for terminal to open, for a device to respond, and so on), the backend is not friendly.

Backend Routines in libqb

The following is a list of backend routines available using the **ld** or **cc** command-line option **-lqb**.

```
burst_page(fnaddr,width)
  int (*fnaddr)();
```

```
get_align()

get_copies()

get_endgroup()

get_feed()

char *
get_from()

get_header()

char *
get_lastuser()

char *
get_moddate()

get_newuser()

char *
get_nodeid()

char *
get_qdate()

char *
get_title()

char *
get_to()

get_trailer

log_charge(charge)
  long charge;
```

```
log_init()

log_message( . . . )

log_progress(pages,percent)

log_status(status)

put_header(fnaddr, width)
  int(*fnaddr)();

put_trailer(user, fnaddr, width)
  char *user;
  int (*fnaddr)();
```

Appendix C. Writing Device Drivers

This appendix explains many of the details involved in writing a device driver for the AIX Operating System. To get the most from this information, you should be familiar with the C programming language, how AIX handles I/O, interrupt-level programming, the Virtual Resource Manager (VRM), the Virtual Machine Interface (VMI), and the interface to the device for which you are writing a driver.

The operating system contains device drivers for all devices known to the system. You can add device drivers to the system to control other devices. Once the device driver is installed, you can use existing commands and utilities.

You can write drivers that allow several processes to use a device at the same time. Since most devices are much slower than the processor, device drivers may also release control of the processor to other processes while waiting for a device to complete an operation.

Device Driver Concepts

When a user program issues a system call that requests input or output, the kernel determines which device driver should handle the data transfer, and calls the proper routine of the device driver to perform the operation. In general, a device driver performs the following operations:

1. Determines whether there is enough buffer space for the transfer, and if not, requests more buffer space from the kernel.
2. Requests the kernel to move the data from the user area of memory to its buffer space.
3. Masks interrupts to avoid being interrupted.
4. Issues an SVC to send a request to the VRM device driver, which communicates the information to the hardware device.
5. Updates data structures to reflect the data that was moved from the user program's buffer.
6. Unmasks the interrupts.
7. Waits for the device to finish the transfer.

When the device finishes the transfer operation, it generates an interrupt, which the VRM passes to the kernel. The kernel then calls the interrupt-handling entry point of the device driver. The device driver interrupt handler ensures that the interrupt was valid, checks the status, and then starts the transfer of data that has been queued.

This cycle continues until the data transfer routine has sent all of the data to the device. When the data transfer is complete, the device driver returns a completion code to the kernel, and the kernel returns the code to the calling program.

An AIX device driver is actually linked, using the **ld** command, into the kernel load module and is therefore part of the kernel.

The device driver can access members of a structure that contains various administrative information about the user process that is called the ***user area***, ***user block***, or ***ublock***. This structure is accessed with the construct **u.***field*, where *field* is a valid field name of **struct user** as defined in the **sys/user.h** header file. For example, the error code stored in **u.u_error** is the value that is passed back to the application program as **errno**.

Warning: Do not:

- Modify any fields of the user block or of any other kernel structure that are not explicitly mentioned in this section
- Call any kernel subroutines that are not explicitly mentioned in this section.

Otherwise, unpredictable and disasterous results may occur.

Types of AIX device drivers

An AIX device driver can be one of two types: ***character*** or ***block***. A block device driver organizes data into fixed-size blocks or clusters that are usually 2048 bytes in size. Each block can be accessed at random when given block number. Block devices include fixed-disk and diskette drives.

A block device driver can perform each transfer as soon as the request is received, or it can queue the requests and perform them in an order that is most efficient for the device being used.

A character device is any device that does not fit the block device definition; the interface for character devices is less structured than the interface for block devices. Character devices include the keyboard, displays, and printers.

Character device drivers can also be designed to provide controlled access to low-level facilities of the system that are not necessarily associated with true I/O devices. Since device driver are accessed by an AIX path name, access to these facilities can be controlled by the access permission mode of the special file (see "chmod" on page 2-18). AIX provides the following special-purpose drivers in addition to others:

tty Allows a program to access its controlling terminal.

null Discards output written to it and indicates an end-of-file condition when read.

bus Permits direct access to the I/O bus for memory-mapped I/O.

mem Provides access to system memory.

kmem Provides access to kernel memory.

config Issues the SVCs that send configuration information to the VRM.

trace Records data when tracing programs.

These and other device drivers that are supplied with the AIX system are described in Chapter 6, "Special Files."

Most block devices also have a character special file and device driver entry points for character data transfer that allow reading and writing unformatted data. For example, the first floppy diskette drive can be accessed as either **/dev/fd0** (block) or **/dev/rfd0** (character). The **r** in the name **/dev/rfd0** stands for ***raw*** because character-oriented access to a block device is called ***raw I/O***.

AIX device driver Entry Points

When a program issues a system call for an I/O device, the kernel issues a call to device driver routines to perform the requested operation.

Each AIX device driver is invoked by the kernel using standard ***entry points***, also called ***interface routines***. Each major device number has a corresponding set of entry points named *dd***init**, *dd***open**, *dd***close**, *dd***ioctl**, *dd***read**, *dd***write**, *dd***strategy**, *dd***print**, and *dd***select**, where *dd* is a prefix that uniquely identifies the device driver.

Not all of these entry points need to be defined for a given driver. For instance, an output-only device such as a parallel printer does not need a *dd***read** routine. Also, *dd***read**, *dd***write**, and *dd***select** apply only to character device drivers; *dd***strategy** and *dd***print** are valid only for block device drivers. In addition, block device drivers usually include *dd***read** and *dd***write** entry points for character-oriented (raw) access to the block device.

The procedure for rebuilding the kernel constructs a table that contains pointers to these entry points for each of the device drivers in the system. This table is called the ***device switch table***, and it is an array of type **struct devsw**, which is defined in the **sys/conf.h** header file. The device switch table is constructed using information from the stanza for each AIX device driver in the **/etc/master** file, which defines the unique *dd* prefix used for each entry point name and lists the entry points that are defined for the device. (The procedure for rebuilding the kernel is explained under "Rebuilding the AIX Kernel" on page C-51.)

In addition to these entry points, a device driver can include a routine to service interrupts that an I/O device generates when an operation is finished. Such an entry point is called a ***second-level interrupt handler (SLIH)***, and, by convention, it is named *dd***intr**. The SLIH is not included in the device switch table; instead, it is identified to the kernel with the **vec_init** kernel subroutine, which is usually called from within the *dd***open** routine. (See "vec_init" on page C-18.)

Entry points for both character and block device drivers:

*dd***init** Called during the AIX start-up procedures to configure the VRM device driver for the device.

*dd***open** Called when the device is opened with an **open** or **creat** system call to get the device ready to transfer data.

*dd***close** Called when the device is to be closed. Puts the device in a known idle state.

*dd***ioctl** Called when a user program invokes the **ioctl** system call. Decodes commands for special functions.

*dd***intr** Called when the I/O device interrupts the main processor.

*dd***print** Called to print device information for debugging or error analysis.

Character and raw device driver entry points:

*dd***read** Called when the user program issues a character device **read**.

*dd***write** Called when the user program issues a character device **write**.

*dd***select** Called when the user program issues a **select** system call to a character device.

Block device driver entry point:

*dd***strategy** Called to schedule a read or write to a block device. Performs block data transfer to or from the device.

Parameters

Some of the parameters passed to the entry points always have the same meaning. These parameters are described here:

devno Major and minor device numbers. This is an **int** that specifies both the major and minor device numbers. For convenience and readability, the **sys/sysmacros.h** header file defines the following macros for manipulating device numbers:

major(*devno***)**	Returns the major device number
minor(*devno***)**	Returns the minor device number
makedev(*maj***,** *min***)**	Constructs a composite device number in the format of *devno* from the major and minor device numbers given.

minor The minor device number.

offchan The read/write character offset or channel ID. This is the same as the value in **u.u_offset**, but declared differently. If the device driver is not multiplexed, then it is an **off_t** value that specifies the character read/write offset, which is also called the ***seek pointer***. If the driver is multiplexed, then it is the **caddr_t** value that was passed to the **setmpx** kernel subroutine identifying the channel. (See "Multiplexed Devices" on page C-20.)

ext The extended system call parameter. The **openx**, **closex**, **readx**, **writex**, and **ioctlx** system calls allow applications to pass an extra, device-specific parameter to the device driver. This parameter is passed to *dd***open**, *dd***close**, *dd***read**, *dd***write**, and *dd***ioctl** as the *ext* parameter. If the application uses one of the nonextended system calls (for example **read**, instead of **readx**), then the *ext* parameter has a value of 0. Using the *ext* parameter is highly discouraged, particularly for **open**, **close**, and **ioctl** operations, because doing so makes a device driver incompatible with much existing software.

Common Entry Points

The following entry points are common to both character and block AIX device drivers.

*dd*init

***dd*init (*devno*, *iodn*, *ilev*, *ddilen*, *ddiptr*)**
int *devno*, *iodn*, *ilev*, *ddilen*;
char **ddiptr*;

The **init** entry point is used to configure the VRM device driver.

Parameters:

devno Major and minor device numbers.

iodn I/O device number, or 0 if not applicable. If *iodn* is -1, then the driver is being deleted and all future attempts to open the driver should fail.

ilev Virtual interrupt level.

ddilen Length of device-dependent information, or 0 if none.

ddiptr Pointer to the device-dependent information, or a **NULL** pointer if none.

The *dd***init** entry point passes initialization information to the device driver. The device driver can attach the device and set the interrupt handler from within *dd***init**, but usually the information is saved in a static structure and is used in *dd***open** to initialize the device.

*dd*open

***dd*open (*minor*, *rwflag*, *ext*)**
int *minor*, *rwflag*, *ext*;

The kernel calls the *dd***open** routine of a device driver when a program issues an **open** or **creat** system call.

Parameters:

minor The minor device number.

flag One of the following values:

FREAD	The device is being opened for reading only.
FWRITE	The device is being opened for writing or reading and writing.

ext The extended system call parameter. Using the *ext* parameter is highly discouraged because doing so makes a device driver incompatible with much existing software.

The *dd***open** entry point prepares a device for reading and writing.

Many character devices, such as printers and plotters, should only be opened by one process at a time. The **open** entry point can enforce this by maintaining a static flag variable, which is set to 1 if the device is open and 0 if not. Each time it is called, *dd***open** checks the value of the flag and, if it is other than zero, sets **u.u_error** to **EIO** and returns a value of -1 to indicate that the device is already open. Otherwise, *dd***open** sets the flag and returns normally. *dd***close** later clears the flag when the device is closed.

Most block devices can be used by several processes at once, and the driver should not usually try to enforce opening by a single user.

Other special processing that is typically done in *dd***open** entry points:

- Define an SLIH (*dd***intr** entry point) by calling the **vec_init** kernel subroutine (see "vec_init" on page C-18).
- Call **usrchar** to get the extended path name of a multiplexed special file and use it to set up a structure containing information about the logical subdevice being opened.
- Define the device driver to be multiplexed by calling **setmpx** (see "Multiplexed Devices" on page C-20).

The *dd***open** entry point can indicate an error condition to the application program by storing a nonzero error code in the **u.u_error** field of the user block. This causes the system call to return a value of -1 and makes the error code available to the application in the **errno** external variable. The error code used should be one of the values defined in the **errno.h** header file, which are also listed under Appendix A, "Error Codes."

*dd*close

*dd***close (***minor*, *offchan*, *ext***)**
int *minor*, *offchan*, *ext*;

The close entry point resets the device to a known state and resets the device controller to prevent it from generating any more interrupts until it is opened again.

Parameters:

minor The minor device number.

offchan The read/write character offset or channel ID.

ext The extended system call parameter. Using the *ext* parameter is highly discouraged because doing so makes a device driver incompatible with much existing software.

If the device is multiplexed, then the kernel calls *dd***close** each time a process issues a **close** system call. If the device is not multiplexed and more than one process has opened it, then the kernel calls *dd***close** only once: when the last process closes it.

The *dd***close** entry point can indicate an error condition to the application program by storing a nonzero error code in the **u.u_error** field of the user block. This causes the system call to return a value of -1 and makes the error code available to the application in the **errno** external variable. The error code used should be one of the values defined in the **errno.h** header file, which are also listed under Appendix A, "Error Codes."

*dd***ioctl**

*dd***ioctl** **(***minor*, *op*, *arg*, *flag*, *offchan*, *ext***)**
int *minor*, *op*, *arg*, *flag*, *offchan*, *ext*;

When a program issues an **ioctl** system call, the kernel calls the *dd***ioctl** routine of the specified device driver.

Parameters:

minor The minor device number.

op The parameter from the system call that specifies the operation to be performed.

arg The parameter from the system call that specifies the address of a parameter block.

flag The file parameter word, which includes bits that indicate whether the file is open for read, write, or append:

FREAD	Open for reading.
FWRITE	Open for writing.
FAPPEND	Open for appending.

offchan The read/write character offset or channel ID.

ext The extended system call parameter. Using the *ext* parameter is highly discouraged because doing so makes a device driver incompatible with much existing software.

See the **/usr/include/sys/file.h** file for a complete list and bit definition of the file parameter word.

Most **ioctl** operations depend on the specific device involved. However, all **ioctl** routines must respond to the following commands:

IOCTYPE Returns a character that indicates the device type.

IOCINFO Returns a structure that describes the device (refer to the description of the special file for a particular device in this book for a description of the structure). Only the first two fields of the data structure need to be returned if the remaining fields of the structure do not apply to the device.

The *dd***ioctl** entry point can indicate an error condition to the application program by storing a nonzero error code in the **u.u_error** field of the user block. This causes the system call to return a value of -1 and makes the error code available to the application in the **errno** external variable. The error code used should be one of the values defined in the **errno.h** header file, which are also listed under Appendix A, "Error Codes."

*dd*intr

*dd***intr (***argp***)**
int ****argp*;**

Warning: The second-level interrupt handler has access to segment 0 only. Attempts to access other segments can produce unpredictable results.

An SLIH or *dd***intr** entry point, is similar in concept to a signal handler (see "signal" on page 2-145). However, a *dd***intr** routine is called in response to an interrupt from an I/O device, which usually indicates that an I/O operation is finished. A *dd***intr** routine also runs in a restricted environment:

- It does not run as a process and, therefore, cannot perform operations that suspend the current process, such as **delay**, **sleep**, and **wait**.
- It can access only segment 0 of memory, the kernel segment. In particular, it cannot access the user block, user data, or the I/O bus.

Due to these restrictions, *dd***intr** routines usually do little more than update a few data structures and call the **wakeup** kernel subroutine to restart the process that actually handles the I/O request.

The *argp* parameter is a pointer to an integer that contains the value given to **vec_init** when it was called in the *dd***init** or *dd***open** routine.

See "Device Interrupts" on page C-18 for more information about interrupts.

Character Device Drivers

*dd*read, *dd*write

*dd***read (***minor*, *offchan*, *ext***)**
int *minor*, *offchan*, *ext*;

*dd***write (***minor*, *offchan*, *ext***)**
int *minor*, *offchan*, *ext*;

When a program issues a **read** system call, the kernel calls the **read** entry point; when a program issues a **write** system call, the kernel calls the **write** entry point.

minor	The minor device number.
offchan	The read/write character offset or channel ID.
ext	The extended system call parameter.

Both entry points use the following system variables to control thè data transfer operation:

u.u_base The address of the beginning of the user buffer relative to the process data segment.

u.u_count The byte count given to the **read** or **write** call.

u.u_offset The file offset established by a previous **lseek**. Most character devices ignore this variable, but some, such as the **/dev/mem** pseudo device, use and maintain it.

Several kernel subroutines are available that automatically update these variables while moving characters to and from the user data space. See "Moving Data for Character I/O" on page C-12 for details about them.

For most devices, the *dd***read** and *dd***write** routines issue an SVC to the VRM and then wait for the VRM to finish. The waiting is accomplished by calling the **sleep** kernel subroutine, which suspends the driver and the process that called it and permits other processes to run. When the I/O operation is completed, the device usually issues an interrupt, causing the device driver's *dd***intr** interrupt handler to be called. *dd***intr** then calls the **wakeup** kernel subroutine to allow the *dd***read** or *dd***write** routine to resume.

When *dd***read** and *dd***write** entry points are provided for raw I/O to a block device, these routines usually translate requests into block I/O requests.

The *dd***read** and *dd***write** entry points can indicate an error condition to the application program by storing a nonzero error code in the **u->u_error** external variable. This causes the system call to return a value of -1 and makes the error code available to the application in the **errno** external variable. The error code used should be one of the values defined in the **errno.h** header file, which are also listed under Appendix A, "Error Codes."

*dd*select

#include <sys/select.h>

int *dd***select (***minor***,** *seltype***,** *chan***)**
int *minor***,** *seltype***;**
caddr_t *chan***;**

Parameters:

minor The minor device number.

seltype One of the following values, specifying the type of selection operation:

FREAD	Read selection
FWRITE	Write selection
0	Exception selection.

chan The channel ID, if this is a multiplexed device.

The *dd***select** entry point is called when the **select** system call is invoked to check whether any data is available to be read, the device is ready to perform a write operation, or an exceptional condition is outstanding for the device.

If the selection criterion specified by the *seltype* parameter is true, then *dd***select** should return a value of 1.

If the selection is not satisifed, then *dd***select** checks to see if another process is already waiting for the same selection criterion to occur. If so, then *dd***select** sets a flag to record the fact that a collision occurred on a select of the requested type. If no other process is waiting, then *dd***select** saves the value of **u.u_procp** to identify the process later, and it returns a value of 0. A separate collision flag and waiting process pointer are kept for each type of **select** request. In effect, this implements a queue with a depth of 1 process waiting for each type of select condition to occur, and sets a collision flag if more than one process is waiting.

If the device is in a state in which the selection criteria can never be satisfied, such as a communication line that is not operational, then *dd***select** should return a 1 so that the process calling **select** does not wait indefinitely.

Note: If your device driver does not support exception selects, then *dd***select** should return a value of 0. If your device driver does not support read or write selects, then *dd***select** should return a value of 1.

For an example of the overall structure of a *dd***select** entry point, see the definition of `ppselect` on page C-42.

The *dd***read** and *dd***write** entry points and the exception-handling routines also require logic to support the **select** operation. Depending on how you write your device driver, your *dd***intr** routine may need to include this logic as well. At each point where one of the selection criteria is true, the device driver checks for a process waiting for that selection and, if one exists, calls the **selwakeup** kernel subroutine to restart it. The collision flag and waiting process pointer that were saved are passed as parameters to **selwakeup**, and then they are reset. For an example of how this can be done, see "A Sample AIX Device Driver" on page C-35.

void selwakeup (*procptr, coll***)**
**struct proc ****procptr*;
int *coll*;

The **selwakeup** kernel subroutine wakes up processes that are waiting on **select**. If the *coll* parameter is a nonzero value, indicating that a collision was detected, then **selwakeup** wakes up all processes waiting on **select**. Otherwise, it wakes up only the process identified by the *procptr* parameter, which is a pointer to the process structure for the requesting process (**u.u_procp**).

Moving Data for Character I/O

Character device drivers can use the following kernel subroutines to transfer data into and out of the user buffer area during **read** and **write** calls. These kernel subroutines use **u.u_base** as the address of the buffer in the user area, and they automatically increment **u.u_base** and **u.u_offset** and decrement **u.u_count** by the number of bytes transferred.

If an invalid user space address is specified, these kernel subroutines set the value of **u.u_error** to **EFAULT**.

You can use these kernel subroutines in multiplexed device drivers even though they update **u.u_offset**. Because seek operations are not allowed on multiplexed files, the kernel does not try to update the read/write character offset of the file.

int cpass ( )

Gets a character from the user buffer that is specified in a **write** system call. Upon successful completion, **cpass** returns the character. If the buffer is empty or if the user base address (**u.u_base**) points to a location outside of the user area, then **cpass** returns a value of -1. In the latter case, **u.u_error** is also set to **EFAULT**.

int passc (*c*)
char *c*;

Stores the character *c* in the user buffer that is specified in a **read** system call. This routine returns a -1 if there is no more space in the user buffer after the character is put into the buffer, or if the base address was not valid. Upon successful completion, **passc** returns a value of 0. If the buffer is full or if the user base address (**u.u_base**) points to a location outside of the user area, then **cpass** returns a value of -1. In the latter case, **u.u_error** is also set to **EFAULT**.

void iomove (*addr*, *count*, *flag*)
char **addr*;
int *count*, *flag*;

Moves a block of data between kernel space and user space. The *addr* parameter points to a buffer in the kernel area, and the *count* parameter specifies the number of bytes to move. The *flag* parameter indicates the direction of the move:

B_READ Copies data from kernel space to user space
B_WRITE Copies from user space to kernel space.

If all or part of the user buffer lies outside of the user area, then **cpass** sets the value of **u.u_error** to **EFAULT**.

Moving Data between User and Kernel Space

The following kernel subroutines do *not* update **u.u_count**, **u.u_offset**, and **u.u_base**. Use them to copy data between user and kernel space.

int subyte (*uaddr*, *c*)
char **uaddr*;
char *c*;

Stores the byte *c* at user data address *uaddr*. **subyte** returns a value of 0 upon successful completion, or -1 if *uaddr* points outside of the user area.

int suword (*uaddr*, *w*)
int **uaddr*;
int *w*;

Stores the word *w* at user data address *uaddr*. **suword** returns a value of 0 upon successful completion, or -1 if *uaddr* points outside of the user area.

int fubyte (*uaddr*)
**char ****uaddr*;

Fetches the byte from user data address *uaddr*. **fubyte** returns the byte upon successful completion, or -1 if *uaddr* points outside of the user area.

int fuword (*uaddr*)
**int ****uaddr*;

Fetches the word from user data address *uaddr*. **fuword** returns the word upon successful completion, or -1 if *uaddr* points outside of the user area. Note that a legitimate value of -1 and the error indication are indistinguishable.

int copyin (*uaddr*, *kaddr*, *count*)
**char ****uaddr*, ******kaddr*;
int *count*;

Copies *count* bytes from user data address *uaddr* to kernel data address *kaddr*. **copyin** returns a value of 0 upon successful completion, or -1 if any or all of the *uaddr* buffer is outside of the user area.

int copyout (*kaddr*, *uaddr*, *count*)

Copies *count* bytes from kernel data address *kaddr* to user data address *uaddr*. **copyout** returns a value of 0 upon successful completion, or -1 if any or all of the *uaddr* buffer is outside of the user area.

Block Device Drivers

Block device drivers must be able to perform multiple block transfers. If the device cannot do multiple block transfers, or can only do multiple block transfers under certain conditions, then the device driver must transfer the data with more than one device operation.

An area of memory is set aside within the kernel memory space for buffering data transfers between a program and the peripheral device. The kernel buffers are allocated in blocks of 2048 bytes. Each block becomes a member of one of the following linked lists that the driver and the kernel maintain:

Available buffer queue
Contains all of the buffers that are not waiting for data to be transferred to or from a device.

Busy buffer queue
Contains all of the buffers that contain data that is waiting to be transferred to or from a device.

Each block in the queue has a **buf** header structure that contains, among other information about the block, two sets of pointers to the next (**forw**) and previous (**back**) members in the list. The device driver maintains these pointers for the available blocks (**av_forw** and **av_back**). The kernel maintains these pointers for the busy blocks (**b_forw** and **b_back**).

The **buf** structure, which is defined in the **sys/buf.h** header file, includes the following fields:

`int`	`b_flags`	Flag bits. See the following discussion.
`struct buf`	`*b_forw`	The forward busy block pointer.
`struct buf`	`*b_back`	The backward busy block pointer.
`struct buf`	`*av_forw`	The forward pointer for a driver request queue.
`struct buf`	`*av_back`	The backward pointer for a driver request queue.
`dev_`	`b_dev`	The major and minor device number.
`unsigned`	`b_bcount`	The byte count for the data transfer.
`caddr_t`	`b_un.b_addr`	The memory address of the data buffer.
`daddr_t`	`b_blkno`	The block number on the device.
`unsigned int`	`b_resid`	Amount of data not transferred after error.

Warning: Do not modify any of the following fields of the **buf** structure that is passed to the *dd***strategy** entry point: **b_forw**, **b_back**, **b_dev**, **b_un**, or **b_blkno**.

Do not modify any of the following fields of a **buf** structure that is acquired with the **geteblk** kernel subroutine: **b_flags**, **b_forw**, **b_back**, **b_dev**, **b_count**, or **b_un**.

Modifying these fields can cause unpredictable and disasterous results.

The value of the **b_flags** field is constructed by logically OR-ing zero or more of the following values:

B_WRITE This is not a read operation.

B_READ This is a read data operation, rather than write.

B_DONE I/O on the buffer has been done, so the buffer information is more current than other versions.

B_ERROR A transfer error occurred, transaction aborted.

B_BUSY The block is not on the free list.

B_PHYS Physical I/O.

B_WANTED A **wakeup** call should be issued when the block is released.

B_AGE The data is not likely to be reused soon, so prefer this buffer for reuse. This flag suggests that the buffer goes at the head of the free list rather rather the end.

B_ASYNC Asynchronous I/O is being performed on this block. When I/O is done, put the block back on the free list rather than waking up a waiting process.

B_DELWRI The contents of this buffer still need to be written out before it can be reused, even though this block may be on the free list. This is used in the **write** routine when the system expects another write to the same block to occur soon.

B_OPEN The **open** routine called.

B_STALE The data conflicts with the data on disk because of an I/O error.

B_DMA **b_un.b_addr** points directly into the user segment.

*dd*strategy

The entry point that performs block-oriented I/O is called ***dd*strategy**:

***dd*strategy (*bufp*)**
**struct buf ****bufp*;

When the kernel needs a block I/O transfer, it calls the strategy routine of the device driver for the device.

The *bufp* parameter points to a **buf** structure. Of particular interest to ***dd*strategy** is the following information:

- The type of transfer (read or write)
- The device block number
- The m‿mory address to be used
- The byte count (number of bytes to be transferred).

This entry point either sends the data to the VRM device driver immediately, or it queues the request for later processing.

void iodone (*bufp*)
**struct buf ****bufp*;

When the block I/O transfer is complete, the device driver must call the **iodone** kernel subroutine to inform the kernel of this fact. The **iodone** kernel subroutine marks the buffer pointed to by the *bufp* parameter to indicate that the I/O has been completed. If the **B_ASYNC** bit of the buffer's **b_flags** field is set, indicating asynchronous I/O, then the buffer is released. Otherwise, **iodone** wakes up any processes that are waiting for the buffer.

Device Interrupts

Interrupts generated by I/O devices are serviced by a second-level interrupt handler, which is an AIX device driver entry point that, by convention, usually has a name of the form *dd***intr**.

The following kernel subroutines define the SLIH to AIX and remove this definition.

level_t vec_init (*level***,** *routine***,** *arg***)**
int *level***;**
int (**routine***) ( );**
int *arg***;**

The **vec_init** kernel subroutine associates an interrupt handler with an interrupt level and sublevel. It is usually called from within the *dd***open** or *dd***init** routine to define the *dd***intr** interrupt handler to the system.

The **vec_init** kernel subroutine associates the interrupt routine pointed to by the *routine* parameter with an unused sublevel for interrupt priority level *level.* The value of the *level* parameter should be that of the *ilev* parameter that was passed to the *dd***init** entry point. This value is always 4 because AIX uses interrupt level 4 for all I/O devices.

After the interrupt handler is installed, a device interrupt associated with this level and sublevel causes *routine* to be called and passed a pointer to an **int** that contains the value *arg*. Thus, the *arg* value can be used to identify the device that caused the interrupt when one AIX device driver controls more than one device.

The **vec_init** kernel subroutine returns a value that identifies the interrupt level and sublevel assigned. The low-order eight bits of this value specify the sublevel, and the next lowest-order three bits specify the level. This is the format used to pass the interrupt level and sublevel to the VRM using the **Attach_Device** SVC.

void vec_clear (*levsublev***)**
level_t *levsublev***;**

The **vec_clear** kernel subroutine removes the association of the interrupt handler with the interrupt level and sublevel specified by the *levsublev* parameter. The value of this parameter is the same as that returned by a previous call to **vec_init**.

Device drivers sometimes need to mask interrupts when in a critical section of code, such as when accessing data that is shared with a *dd***intr** interrupt handler. The following kernel subroutines mask and unmask interrupts:

int splx (*level***)**
int *level*;

> The **splx** kernel subroutine masks interrupts on the level specified by the *level* parameter, and then returns the current level. Because all interrupts occur on the level 4, a value of 0 disables interrupts and a value other than 0 enables interrupts.

Alternatively, you can use the following kernel subroutines:

int spl0 ( )	Enables all interrupts and returns the current level
int spl4 ( )	Disables all interrupts and returns the current level
int spl5 ( )	Disables all interrupts and returns the current level
int spl6 ( )	Disables all interrupts and returns the current level
int spl7 ( )	Disables all interrupts and returns the current level
int splhi ( )	Disables all interrupts and returns the current level.

Multiplexed Devices

A multiplexed device is one that uses the following kernel subroutines to decode special information that is appended to the end of the path name of the special file for the device. This path name extension is frequently used to identify a logical or virtual subdevice, or ***channel***. The **usrchar** kernel subroutine provides access to the path name extension, and **setmpx** provides a means by which individual channels can be recognized on subsequent I/O calls.

char usrchar ( )

Each call to **usrchar** returns a character from the path name of a multiplexed special file, starting immediately after the / (slash) that ends the system file name. At the end of the file name, a value of 0 ('\0') is returned. If the path name has no more characters after the special file name, a value of 0 is returned on the first call.

The **usrchar** kernel subroutine is normally called from the *dd***open** routine to determine the channel of a multiplexed device that is being opened.

int setmpx (*chan***)**
caddr_t *mpx***;**

The **setmpx** kernel subroutine is called from the *dd***open** routine to declare a device driver to be multiplexed. The value of the *chan* parameter is saved in the open file table for the process, and it is passed to the *dd***read**, *dd***write**, *dd***ioctl**, and *dd***close** routines to identify this instance of *dd***open**.

Note that the **lseek** system call is not allowed on a multiplexed file, and that the *dd***close** entry point is called each time a process closes it. (For nonmultiplexed files, *dd***close** is called only once: when the last process that opened it closes it.)

Process Suspension and Timing

The operating system provides the following kernel subroutines to suspend and synchronize processes:

sleep (*chan*, *pri*)
int *chan*, *pri*;

Deactivates the calling process on channel *chan*. When the process activates again, it runs with the priority specified by *pri*. The new priority is effective only while the device driver has control. Once it returns to the user program, the kernel controls the priorities.

All processes that are waiting on the channel are restarted at once, causing a race condition to occur. Thus, after returning from the **sleep** kernel subroutine, each process should check to see whether it needs to **sleep** again.

The channel specified by the *chan* parameter is simply a value that identifies an event to wait for, or to sleep on. This value is passed to the **wakeup** kernel subroutine to start up all of the processes that are waiting for the event. The channel identifier must be unique system-wide, so the address of an external kernel variable (which can be defined in a device driver) is generally used for this value.

wakeup (*chan*)
int *chan*;

Makes all processes that were suspended on channel *chan* by the **sleep** kernel subroutine ready to execute. The processes do not actually begin to execute until the current process relinquishes control of the processor or returns to user mode. Because all processes that are waiting on the channel are restarted, a race condition occurs. Thus, after returning from the **sleep** kernel subroutine, each process should check to see whether it needs to **sleep** again.

The channel specified by the *chan* parameter is simply a value that identifies an event to wait for, or to sleep on. This value is passed to the **wakeup** kernel subroutine to start up all of the processes that are waiting for the event. The channel identifier must be unique system-wide, so the address of an external kernel variable (which can be defined in a device driver) is generally used for this value.

The **wakeup** kernel subroutine is usually called from an interrupt handler (*dd***intr**).

timeout (*func*, *arg*, *ticks*)
int (**func*) **( );**
int *arg*, *ticks*;

Schedules the function pointed to by the *func* parameter to be called with the parameter *arg* after *ticks* timer ticks. Timer ticks occur many times a second, but they do not necessarily occur at regular real-time intervals because they are caused by virtual interrupts from the VRM.

The *func* function is called on an interrupt level; therefore, it must follow the conventions for interrupt handlers. See "*dd***intr**" on page C-9 for more information about the restrictions that apply to interrupt handlers.

untimeout (*func***,** *arg***)**
int (**func***) ( );**
int *arg***;**

Cancels all pending **timeout** requests that were made by calling **timeout** with the specified *func* and *arg* values.

delay (*ticks***)**
int *ticks***;**

Suspends the calling process for the number of timer ticks specified by the *ticks* parameter. This is implemented by using **timeout** to schedule a call to **wakeup,** then calling **sleep** to wait for the **wakeup.**

Processes have two kinds of sleep: ***fast*** and ***slow***. When the wakeup priority is numerically less than or equal to the **PZERO** value defined in **/usr/include/sys/pri.h,** sleep is fast. In this case, no signal or other external action can interrupt the process.

If the priority is numerically greater than **PZERO**, then the sleep is slow. In this case, a signal such as one caused by the **kill** or **alarm** system calls, or pressing the **Alt-Pause** key sequence aborts the sleep and restarts the process. When this happens, the process does not return from the **sleep**, but uses **longjmp** to return to the address saved in **u.u_qsav**. By default, this causes the system call to return a value of -1 to the user process and to set **errno (u.u_error)** to **EINTR**, indicating that the system call was interrupted by a signal.

Device drivers can take either of two approaches to handling signals. The simpler and more common approach is to write the driver so that it will continue to operate correctly even if the **sleep** routine does not return. This approach is taken in the sample driver shown in "A Sample AIX Device Driver" on page C-35.

The second approach is that instead of letting the process return to the address saved in **u.u_qsav**, the process can catch and process the signal. To do this, logically OR the value **PCATCH** with the *pri* parameter passed to the **sleep** routine. If the **sleep** kernel subroutine returns due to a signal, then it returns a nonzero value. Otherwise, it returns a value of 0.

This second approach is demonstrated in the following example.

```
if ( sleep(chan, PZERO + 4 | PCATCH) == 1 )
{
   /* ... Perform operations in response to the signal ... */
   u.u_error = EINTR;   /* Indicate the interruption       */
   return;
}
```

In this case, a signal does not restart the process, but returns control to the next sequential instruction after the call to **sleep**. This allows the kernel code to perform operations in response to the signal before returning normally.

Setting the **runrun** external variable to a nonzero value causes the scheduler to dispatch the next procedure with the highest priority at the next opportunity. Such opportunities occur when returning from an interrupt handler, leaving kernel mode, and returning from a system call. You can set **runrun** to cause a signal to preempt the process that is currently running. To set **runrun**, use the following statement:

```
runrun++;
```

Interprocess Communication

The following kernel subroutines allow an AIX device driver to communicate directly with user processes.

Signals

psignal (*p*, *sig*)
struct proc **p*;
int *sig*;

The **psignal** kernel subroutine sends a signal to a process. The *p* parameter points to the process table entry for the receiving process. The *sig* parameter specifies the signal to send.

To get the value for the *p* parameter, save **u.u_procp**, which contains a pointer to the process table entry for the process that made the system call.

See "signal" on page 2-145 for a list of the valid signals and "sigvec" on page 2-156 for more information about how signals work.

Message Queues

The following kernel subroutines provide the same message queue services as the **msgctl**, **msgget**, **msgsnd**, and **msgxrcv** system calls. The return values are the same, and the error code stored in **errno** can be accessed by the device driver as **u.u_error**. Note that a memory fault (**EFAULT**) cannot occur because message buffer pointers in the kernel address space are assumed to be valid.

These subroutines must be called from the process level, not the interrupt level (*dd***intr**), since they call **sleep** when waiting for resources.

int kmsgctl (*msqid*, *cmd*, *buf*)
int *msqid*, *cmd*;
struct msqid_ds *buf*;

See the description of "msgctl" on page 2-73.

int kmsgget (*key*, *msgflg*)
key_t *key*;
int *msgflg*;

See the description of "msgget" on page 2-76.

int rmsgsnd (*msqid***,** *msgp***,** *msgsz***,** *msgflg***,** *segflg***)**
int *msqid***;**
struct msgbuf ****msgp*;**
int *msgsz***,** *msgflg***,** *segflg***;**

The *msqid*, *msgp*, *msgsz*, and *msgflg* parameters are described under "msgsnd" on page 2-82. The *segflg* parameter should be 1, indicating that the **msgbuf** is in the kernel address space, not user space.

int rmsgrcv (*msqid***,** *msgp***,** *msgsz***,** *msgtyp***,** *segflg***)**
int *msqid***;**
struct msgxbuf ****msgp*;**
int *msgsz***;**
mtype_t *msgtyp***;**
int *segflg***;**

The *msqid*, *msgp*, *msgsz*, and *msgtyp*, parameters are described under "msgxrcv" on page 2-85. The *segflg* parameter should be **(XMSG | 1)**, where **XMSG** indicates that this is an extended message receive, and the 1 indicates that the **msgxbuf** is in the kernel address space, not user space.

Dynamic Storage Management

If a device driver expects to buffer more than a few characters of data, it should use the dynamic storage facilities provided by the kernel. The following sections discuss three separate facilities.

malloc, palloc, free

The **malloc**, **palloc**, and **free** routines describe allocation of kernel address space and provide a simple interface for kernel address space allocation.

caddr_t malloc (*size*)

The **malloc** kernel subroutine returns a pointer to an area that is *size* bytes in length. It returns a **NULL** pointer if the requested block of memory cannot be allocated.

caddr_t palloc (*size*, *align*)

The **palloc** kernel subroutine returns a pointer to an area of length *size* aligned on an address boundary of 2 raised to the power specified by the *align* parameter (2^{align}). For example, `palloc(1024, 11)` returns a pointer to a 1024-byte area that is aligned on a 2048-byte boundary ($2^{11} = 2048$). **palloc** returns a **NULL** pointer if the requested block of memory cannot be allocated.

void free (*ptr*)

free returns an area allocated by **malloc** or **palloc** to the free buffer pool.

malloc and **palloc** can only be called from process level. **free** can be called from process level or interrupt level.

Character Lists

Character lists are used to maintain a queue of bytes, which can also be treated as a queue of buffers that contain groups or blocks of bytes. They are appropriate for relatively slow devices such as terminals and printers.

For each character list that you use, you must define a queue header, which is a variable of type **struct clist**. This **clist** structure is defined in the **sys/tty.h** header file, and it contains the following members:

int	c_cc	Character count
struct cblock	*c_cf	Pointer to first block
struct cblock	*c_cl	Pointer to last block

You do not need to be concerned with maintaining the fields in the **clist** header; the character list kernel subroutines do that for you.

Each block in the queue is a **cblock** structure, which is also defined in the **sys/tty.h** header file:

struct cblock	*c_next	Pointer to next block
char	c_data[CLSIZE]	Data
char	c_first	Offset to first character
char	c_last	Offset to last character

A block does not need to be completely filled with characters. The fields **c_first** and **c_last** are zero-based offsets within the **c_data** array, which actually contains the data.

The following kernel subroutines are provided for handling character lists. All of the routines mask the interrupts as needed, so you can call them from either mainline or interrupt level.

int getc (*header*)
struct clist *_header_;

Returns the next character from the queue whose header is pointed to by the *header* parameter. If it is the last character in the buffer, the buffer is freed. If the buffer is empty, then **getc** returns -1.

int putc (*c*, *header*)
char *c*;
struct clist *_header_;

Puts the character *c* in the queue whose header is pointed to by the *header* parameter. If the operation is successful, a value of 0 is returned. If the queue is full and no more blocks are available, then **putc** returns -1.

struct cblock *getcf ( )

Gets a block from the free list and returns a pointer to it. If no blocks are available, then **getcf** returns a **NULL** pointer. If you get buffer space with this routine, you must ensure that you free the space when you are through with it.

void putcf (*p*)
struct cblock ****p*;**

Returns the block specified by the *p* parameter to the free block list.

struct cblock *getcb (*header*)
struct clist ****header*;**

Returns the address of the next buffer on the queue specified by the *header* parameter, or a **NULL** pointer if the queue is empty. The buffer is removed from the queue as well. If you get a buffer with this routine, you must ensure that you free the space when you are through with it.

void putcb (*p*, *header*)
struct cblock ****p*;**
struct clist ****header*;**

Puts the buffer pointed to by *p* on the queue specified by *header*. Befure calling **putcb**, you must load this new buffer with characters and set **c_first** and **c_last**. The *p* parameter is the pointer returned by either **getcf** or **getcb**.

You can mix calls to **getc**, **putc**, **getcb** and **putcb**. In this manner, you can insert characters in the buffer one by one, and remove them as a group. You can also insert them as a group and remove them one by one.

The amount of system memory available for character queues is limited. All character device drivers must share this pool of memory. Therefore, you must limit the number of characters in your queue space to a few hundred. When the device is closed, the driver should make certain that all of its character queues are flushed so that the character blocks are returned to the system.

Disk Buffer Header Allocation

Device drivers (even character device drivers) can get buffers from the system supply of available disk buffers. The use of disk buffers by character device drivers is strongly discouraged. Instead, using **malloc** to allocate memory space is preferred. Because these buffers cannot then be used to buffer disk I/O, be careful to get only the amount of space needed.

"Block Device Drivers" on page C-15 explains the fields of the buffer structure and describes how these buffers are maintained. In the header, the **b_forw, b_back, b_flags, b_bcount**, **b_dev** and **b_un** fields are used by the system and may not be modified by the driver. The **av_forw** and **av_back** fields are available for keeping a chain of such buffers by the kernel or by the driver. **b_blkno** and **b_resid** can be used for any purpose. If input or output is to occur directly to the user address space (sometimes referred to physical I/O), the **b_proc** entry contains a pointer to the process table entry for the process performing a read or write. Segment register information is stored in the process structure, therefore, the **b_proc** entry is used in an interrupt routine that starts a physical I/O to ensure that the segment is actually mapped into memory when the start I/O SVC is issued.

Two kernel subroutines allow you to get and release buffers for use by your device driver. When disk buffers are used in a device driver, the *dd***open** entry point typically calls **geteblk** to get the buffers, and *dd***close** calls **brelse** to return them.

struct buf *geteblk ( )

Returns the address of a buffer header that is not in use. Note that **geteblk** does *not* allocate disk data buffers. If no free buffer headers are available, then **geteblk** waits for one to become available. Therefore, you can call this routine only from the process level, not from the interrupt level (*dd***intr**).

void brelse (*b*)
struct buf **b*;

Frees the buffer that is pointed to by the *b* parameter. This kernel subroutine can be called from either the interrupt level or the process level.

Block Buffer Allocation

Some device drivers may need to use the Block I/O Communication Area (BIOCA). The BIOCA is a block of storage in the kernel address space that is allocated when the device driver is configured. The **/etc/biohelp** customize helper can be used to configure a device driver that uses the BIOCA. See the discussion of IBM device drivers in *Virtual Resource Manager Technical Reference* for detailed information about the BIOCA, and see *AIX Operating System Programming Tools and Interfaces* about configuring device drivers.

A device driver can request and release buffers from the BIOCA with the following kernel subroutines:

#include <sys/bioca.h>

struct biobuf *bfget (*poolptr*)
char **poolptr*;

The **bfget** kernel subroutine returns a pointer to the next available buffer in the BIOCA. If no buffers are available, then **bfget** returns a **NULL** pointer. The *poolptr* parameter is a pointer to the buffer pool that is passed back from the block I/O manager.

#include <sys/bioca.h>

int bffree (*bufaddr***)**
struct biobuf ****bufaddr*;**

The **bffree** kernel subroutine frees the buffer pointed to by the *bufaddr* parameter. **bffree** always returns a value of 0.

Error Logging and Console Messages

void printf (*format* [, *value*, . . .]**)**
**char ****format*;

The **printf** kernel subroutine writes a formatted character string to the error log (using the **errsave** kernel subroutine), and then it writes the string to the console. Most of the system's operation is suspended while **printf** is writing to the console, so use it only for important messages.

The **printf** kernel subroutine resembles the **printf** subroutine described on page 3-300, but do not confuse the two. The subroutine is part of the Standard I/O Package (**libc.a**), which is used by application programs. The kernel subroutine is built into the kernel and is accessible only within the kernel and AIX device drivers. Also, the **printf** kernel subroutine recognizes only the `%s`, `%d`, `%D`, `%o`, and `%x` conversion specifications. Field width, precision, and other modifiers are not recognized.

The record written to the error log is marked as being an informational message about an AIX device driver. The fields of the error log header are set to the following values, as defined in the **sys/erec.h** header file:

class	**E_SOFTWARE**
subclass	**E_UNIX**
mask	**E_UNIXDD1**
type	**E_INFO**

int errprintf (*format* [, *value*, . . .]**)**
**char ****format*;

The **errprintf** kernel subroutine is identical to **printf**, except that it writes the formatted string to the console only if the error device driver, **/dev/error**, has not been opened.

panic (*s***)**
**char ****s*;

The **panic** kernel subroutine is called when a catastrophic error occurs and the system can no longer continue to operate. It performs the following actions:

- Uses **printf** to write the character string pointed to by the *s* parameter to the console, preceded by the word `panic:`
- Performs a **sync** operation, flushing all disk buffers
- Does a system dump
- Records the first 14 characters of the **s** string in non-volatile random access memory (NVRAM)
- Goes into an endless idle loop during which interrupts are processed.

If **panic** is called a second time while processing an interrupt, then **panic** performs the following actions:

- Writes the *s* string to the console, preceded by the phrase `Double panic:` **(printf)**
- Records the first 14 characters of the new **s** string in NVRAM
- Goes into an endless idle loop during which interrupts are processed.

void errsave (*buf***,** *cnt***)**
char ****buf*;**
unsigned int *cnt***;**

The **errsave** kernel subroutine allows AIX device drivers and the AIX kernel to write error log entries to the error device driver. Application programs should use the **errunix** kernel subroutine to log error messages.

The *buf* parameter points to a buffer that contains the following information:

1. A word (**int**) that contains the **class**, **subclass**, **mask**, and **type** of the message, as defined in the discussion of "error" on page 6-15

2. A word that specifies the number of words of dependent data for the error log entry, including this **int** itself

3. Dependent information for the error log entry. The number of dependent data words must be one less than the word count specified immediately before them.

The other fields of the error log header (length, date and time, time extended, node name, and virtual machine ID) are supplied for you automatically.

The *cnt* parameter specifies the number of bytes in the buffer pointed to by *buf*. The value of *cnt* must be a multiple of 4.

See also "errunix" on page 3-126, and "error" on page 6-15.

Trace Logging

void trsave (*traceid*, *cnt*, *buf***)**
unsigned short *traceid***;**
char ****buf*;**
unsigned int *cnt***;**

The **trsave** kernel subroutine allows AIX device drivers and the AIX kernel to write trace log entries to the trace device driver. Application programs should use the **trcunix** kernel subroutine to log trace events.

The high-order 5 bits of the *traceid* parameter specify the channel number, and the low-order 11 bits specify the *hookid* for the message. User programs may use only channel number 31. The *buf* parameter points to a buffer that contains up to 20 bytes of data for the trace log entry. The *cnt* parameter specifies the number of bytes in the buffer pointed to by *buf*.

If the system trace device driver has already been opened, and if the channel specified by the *traceid* parameter has been enabled, then the driver stores the log entry in a queue. If there is not enough room in the queue, then the entire entry is discarded and a special entry is made to record the fact that it was discarded.

See also "trace_on" on page 3-357, "trcunix" on page 3-362, and "trace" on page 6-128.

VMI Supervisor Calls

Services are requested of the VRM by using VMI supervisor calls (SVCs). The following kernel subroutines issue these SVCs for AIX device drivers. The AIX device driver must include the **sys/ksvc.h** header file in order to use these routines.

Supervisor Call	Description
S_ST (*a*, *b*, *c*)	Set Timer
S_SI (*a*)	Set Interrupt
S_SOI (*a*, *b*, *c*, *d*, *e*)	Soft Interrupt
S_RFI ( )	Return From Interrupt
S_VMW ( )	VM Wait
S_D ( )	Dispatch
S_R ( )	Return
S_DEFDEV (*a*)	Define Device
S_XADEV (*a*, *b*, *c*)	Attach Device
S_XDDEV (*a*, *b*)	Detach Device
S_QDEV (*a*)	Query Device
S_XSIO (*a*, *b*)	Start IO
S_CIO (*a*, *b*, *c*)	Cancel IO
S_CRSEG (*a*)	Create Segment
S_DSEG (*a*)	Destroy Segment
S_LCSEG (*a*)	Logical Copy Segment
S_CSSEG (*a*, *b*)	Change Size of Segment
S_PP (*a*, *b*, *c*, *d*)	Protect Pages
S_QPP (*a*)	Query Page Protection
S_LSR (*a*)	Load Segment Registers
S_CSR (*a*)	Clear Segment Registers
S_PINPR (*a*, *b*, *c*)	Pin Page Range
S_UPR (*a*, *b*, *c*)	Unpin Page Range
S_PURPR (*a*, *b*, *c*, *d*, *e*)	Purge Page Range
S_SIM (*a*, *b*, *c*, *d*)	Send Immediate Message
S_SAM (*a*, *b*, *c*, *d*)	Send Address Message
S_SMR (*a*)	Set Message Receive
S_XTVM (*a*)	Terminate Virtual Machine
S_DCODE (*a*, *b*, *c*, *d*)	Define Code
S_NOOP ( )	No-op; sets hardware floating-point processor registers
S_KOUTPUT (*a*, *b*, *c*)	KSR Output
S_KSO (*a*, *b*, *c*, *d*, *e*, *f*)	KSR Short Output
S_KSS (*a*, *b*, *c*, *d*)	Set Structure

Supervisor Call	Description
S_KSQ (*a*, *b*, *c*, *d*, *e***)**	KSR Struct Query
S_MAP (*a***)**	Map Page Range
S_UNSHRMPR (*a*, *b*, *c***)**	Unshare Mapped Page Range
S_XSC (*a*, *b*, *c*, *d*, *e*, *f***)**	Send Command
S_IPLVM (*a*, *b*, *c***)**	IPL Virtual Machine
S_UPDVRM ( )	Update VRM
S_VIPL ( )	Re-IPL VRM
S_POST (*a*, *b***)**	Post
S_RQGET (*a***)**	Ring Queue Get
S_RQPUT (*a*, *b***)**	Ring Queue Put
S_QVM (*a*, *b*, *c***)**	Query Virtual Machine
S_RNVRAM (*a*, *b*, *c***)**	Read NVRAM
S_WNVRAM (*a*, *b*, *c***)**	Write NVRAM
S_ALLOCFP (*a*, *b***)**	Alloc FP hardware register sets
S_FREEFP (*a***)**	Free FP hardware register sets
S_QUERYFP ( )	Query FP hardware register sets

A Sample AIX Device Driver

The following example is a complete character device driver. This driver does the basic operations necessary to run the device, but little or no error checking is done. A more robust driver would examine the return value of each SVC to determine success or failure. Also, the interrupt handler should test the operation result in the interrupt PSB to determine success or failure. See *Virtual Resource Manager Technical Reference* for discussions of SVC error codes and formats of the interrupt PSB.

```
/*   Example device driver program for device
     similar to parallel printer                */

#include <sys/param.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/tty.h>
#include <sys/devinfo.h>
#include <sys/errno.h>
#include <sys/kio.h>
#include <sys/select.h>

#define DD_PP   'P'                  /* this would be defined in your
                                        devinfo.h            */
#define PP_WRITE  0x1                /* ccb operation */
#define PPPRI   (PZERO+8)            /* sleep priority */
#define PPLOWAT 40                   /* try not let queued below this */
#define PPHIWAT 100                  /* try not let queued above this */
#define OPEN    0x01                 /* already open flag */
#define ASLEEP  0x08                 /* asleep flag */
#define HOG     0x10                 /* queued chars gone above HIWAT */
#define WCLOSE  0x20                 /* close pending */
#define PPBUSY  0x40                 /* io svc in progress */
#define PPWCOL  0x80                 /* write select collision */

/*     macros for convenience         */
#define min(x, y)      (((x) < (y)) ? (x) : (y))

/*     validate range of minor device number */
#define DEVCHECK(dev) \
        pp = &pp_dev[minor(dev)]; \
        if (pp > &pp_dev[PPMAX]) \
        { \
           u.u_error = ENXIO; \
           return; \
        }

/*      device table entry      */
struct pp {
        int            p_pvec;     /* interrupt level, sublevel */
        unsigned short p_iodn;     /* iodn of vrm device */
        struct  clist  p_outq;     /* head of clist */
        struct  cblock *p_block;   /* current cblock */
        long           p_pathid;   /* path id of vrm device */
        struct ccb     p_ccb;      /* ccb for start io */
        struct com_elm p_elm;      /* element for start io */
        char           p_state;    /* state flags */
        struct proc    *p_selw;    /* proc waiting on select */
        short          p_ilevel;   /* interrupt level */
};

/*      device table    */
#define PPMAX  2
struct pp pp_dev[PPMAX] ;

char lbuf[PPHIWAT-PPLOWAT+1];      /* temp copy buffer */

/*      driver open routine     */
ppopen(dev, mode)
dev_t dev;
{
        struct pp *pp;
        int ppintr;

                        /* validate device number */
        DEVCHECK(dev);

                        /* if already open, don't allow another  */
        if (pp->p_state & OPEN) {
                u.u_error = EBUSY;
                return;
        }

                        /* if not already assigned, assign interrupt
                           level on 4 */
        if (pp->p_pvec == 0)  pp->p_pvec = vec_init(4, ppintr, pp);

                        /* perform attach  SVC     */
        if (S_XADEV(pp->p_pvec << 16 | pp->p_iodn, 5, &pp->p_pathid) != 0)
        {       u.u_error = EIO;
                return;
        }

                        /* mark as now open        */
        pp->p_state |=  OPEN ;
}

/*      driver close routine    */
ppclose(dev)
{
        struct pp *pp;

        DEVCHECK(dev);
                        /* mask interrupts */
        spl4();
                        /* flush remaining chars */
        ppkick(pp);
                        /* wait for I/O complete */
        while (pp->p_state & PPBUSY) {
                pp->p_state |= (ASLEEP | WCLOSE);
                sleep(&pp->p_block, PPPRI);
        }
                        /* unmask interrupts */
        spl0();
                        /* detach device */
        S_XDDEV(pp->p_iodn, pp->p_pathid);
        pp->p_state = 0;
}

/*      driver write routine    */
ppwrite(dev)
{
        char *p;
        int n;
        struct pp *pp;

        DEVCHECK(dev);
                  /* while output data still remains */
        while (u.u_count) {
                          /* if number queued below high water mark */
                if (PPHIWAT < pp->p_outq.c_cc) {
                                  /* mask interrupts */
                    spl4();
                                  /* get io going */
                    ppkick(pp);
                                  /* stay asleep if queued characters
                                     still above high water threshold */
                    while (PPHIWAT < pp->p_outq.c_cc) {
                            pp->p_state |= ASLEEP | HOG;
                                sleep(&pp->p_state, PPPRI);
                    }
                                  /* unmask interrupts */
                    spl0();
                }
                          /* copy another chunk */
                p = lbuf;
                n = min(u.u_count, sizeof lbuf);
                if (copyin(u.u_base, p, n))
                {       u.u_error = EFAULT;
                        return;
                }
                u.u_base += n;
                u.u_count -= n;
                                /* put on clist */
                while (n--)
                        while (putc(*p++, &pp->p_outq)) {
                                delay(4);
                        }
        }
}

/*      driver ioctl routine    */
ppioctl(dev, cmd, arg, mode)
int dev;
int cmd;
caddr_t arg;
{
        struct pp *pp;
        struct devinfo dinfo;

        DEVCHECK(dev);
                        /* perform standard info ioctls only */
        switch(cmd) {
        case IOCTYPE:
                u.u_rval1 = DD_PP<<8;
                break;
        case IOCINFO:
                dinfo.devtype = DD_PP;
                dinfo.flags = 0;
                if (copyout((char *)&dinfo, arg, sizeof(dinfo)))
                        u.u_error = EFAULT;
                break;
        default:
                u.u_error = EINVAL;
        }
}

/*      driver init routine */
ppinit(dev,iodn,ilev,ddlen,ddptr)
dev_t dev;
ushort iodn;
short ilev;
short ddlen;
char *ddptr;
{
        struct pp *pp;

        DEVCHECK(dev);
        pp->p_ilevel = ilev;
        pp->p_iodn = iodn;
        return(0);
}

/*      routine to start output and keep it going       */
/*      called from both base level and interrupt level */
/*      always called with interrupts masked            */
ppkick(pp)
struct pp *pp;
{
        struct ccb *lb;
        struct com_elm *le;
        struct cblock *lcb;

                        /* if busy, then interrupt pending.
                           no kick needed */
        if (pp->p_state & PPBUSY) return;

                        /* see if characters still queued for output */
        if ( !(pp->p_block = getcb(&pp->p_outq)) ) {
                /* No more to do.
                   See if a close is pending and wake up process. */
           if (pp->p_state & WCLOSE) {
                   pp->p_state &= ~(ASLEEP | WCLOSE);
                   wakeup(&pp->p_block);
           }
                /* otherwise get next block of characters sent */
        } else {
           pp->p_state |= PPBUSY;
                           /* -> to I/O element    */
           le = &pp->p_elm;
```

```
                           /* -> to ccb           */
           lb = &pp->p_ccb;
                           /* -> to data to output */
           lcb = pp->p_block;
                           /* address of data */
           le->memaddr = lcb->c_data + lcb->c_first;
                           /* number of characters */
           le->tranlen = lcb->c_last - lcb->c_first;
                           /* last element, none to follow */
           le->linkwd = 0;
                           /* iodn of device to receive data */
           lb->iodn = pp->p_iodn;
                           /* interrupt on completion,
                              interrupt on error, element follows
                              write command */
           lb->dev_opt = CCB_INTCOMP | CCB_INTERR | CCB_ELM | PP_WRITE;
                           /* no auto line feed */
           lb->dd.pr.afpai = 0;
           S_XSIO(lb, pp->p_pathid);       /* Start I/O */
        }
               /* when number of pending characters falls below low
                  threshold wake up process      */
        if ((pp->p_state & HOG) != 0 && pp->p_outq.c_cc <= PPLOWAT) {
                pp->p_state &= ~(ASLEEP | HOG);
                wakeup(&pp->p_state);
                if (pp->p_selw)
                {
                   /* Wake up the process that is waiting on a */
                   /*  write select.                           */
                   selwakeup(pp->p_selw,
                             pp->p_state & PPWCOL);
                   pp->p_selw = NULL;
                   pp->p_state &= ~PPWCOL;
                }
        }
}
```

```
/* interrupt handler */
ppintr(pdev)
struct pp **pdev;       /* specified via vec_init */
{
        /* note that some checking for success should be performed by
        interrogating the PSB operation result                    */

                        /* get device pointer */
        struct pp *pp = *pdev;
                        /* no interrupt pending */
        pp->p_state &= ~PPBUSY;
        if (pp->p_block) {
                        /* give back cblock */
                putcf(pp->p_block);
                pp->p_block = 0;
        }
                        /* mask interrupts */
        spl4();
                        /* get next block started */
        ppkick(pp);
                        /* unmask interrupts */
        spl0();
}

ppselect(dev, seltype, offchan)
int dev, seltype;
caddr_t offchan;
{
   int rc, s;
   struct pp *pp;

   DEVCHECK(dev);
   if ( /* Selection criterion can never be satisfied */ )
   {
      rc = 1;
   }
   else
   {
```

```
rc = 0;
/* Mask interrupts while checking device status.          */
s = spl5();

switch (seltype)
{
   case FWRITE:
      if (pp->p_outq.c_cc <= PPHIWAT)
      {
         /* Queued chars are below high-water threshold */
         rc = 1;
      }
      else
      {
         /* Selection criterion not met; return 0.        */
         rc = 0;

         if (pp->p_selw &&
             pp->p_selw->p_wchan == &selwait)
         {
            /* Another process is already waiting for      */
            /*  a write select on this device.  Therefore,  */
            /*  set the device write select collision flag. */
            pp->p_state |= PPWCOL;
         }
         else
         {
            /* Save process information since the select   */
            /*  system call may put this process to sleep. */
            pp->p_selw = u.u_procp;
         }
      }
      break; /* FWRITE */

   case FREAD:
      /* This device driver can always accept reads.      */
      rc = 1;
      break;
```

```
         case 0: /* Exception selection */
            /* This device driver has no exceptions.            */
            rc = 0;
            break;

      } /* switch */

      /* Reset interrupt mask.                                  */
      splx(s);
   } /* if */

   return(rc);
}
```

Installing Device Drivers

After a device driver is written, it should be installed on the system. This section discusses installing device drivers in the AIX kernel. It also provides an example of a helper program that builds the VRM structure to support the AIX kernel driver. This section also lists the steps to rebuild an AIX kernel.

Device drivers are loaded when the system is booted. After it is loaded, the device driver becomes part of the kernel that provides access to the device it controls. VRM device drivers are defined and bound to the AIX kernel device drivers by a combination of the **/etc/vrmconfig** program and the **customize helper** programs. This process takes place primarily at system power-on (IPL). The **/etc/vrmconfig** program uses information in the **/etc/system** and **/etc/master** files and to issue the **Define_Code** supervisor call (SVC) to the VRM to define device driver code in the VRM. Another step required to make a VRM device driver operational is to issue the **Define_Device** SVC to the VRM, passing a pointer to a buffer containing the Define Device Structure (DDS) . The customize helper issues this second VRM SVC as well as binding this new VRM device driver to the corresponding AIX kernel device driver. See *Virtual Resource Manager Technical Reference* for additional information about binding VRM device drivers. The **Define_Code** and the **Define_Device** SVCs use the **ioctl** system calls in the configuration kernel device driver, **/dev/config**.

IBM-supplied VRM device drivers use **/etc/vrcmain** as a customize helper. The keyword **config** = *name* in kernel device driver stanza of the **/etc/master** file identifies the customize helper program used by the device driver. For IBM-supplied software this is specified as **config** = **vrcmain**. The **/etc/vrmconfig** program calls the customize helper program passing the parameters that relate to a stanza in the **/etc/system** being processed. When writing your own VRM and AIX kernel device drivers, you must also provide a customize helper program. The information that follows describes the relationships and other details.

Customize Files

Several files are used to configure the AIX Operating System. These files include:

- /etc/system (see "system" on page 4-139)
- /etc/master (see "master" on page 4-98)
- /etc/ddi/*filename* (see "ddi" on page 4-43)
- /etc/filesystems (see "filesystems" on page 4-64)
- /etc/ports (see "ports" on page 4-117)
- /etc/rc (see *AIX Operating System Commands Reference*)

The **/etc/vrmconfig** and customize helper programs only use **/etc/system, /etc/master** and the files in the **/etc/ddi** directory. The **/etc/system** file is the focal point for all

RT PC customize information. This file contains contains the information in stanzas. See "attributes" on page 4-20 for a detailed description of stanzas.

Customize File Format

In general, a customize file has the format:

```
stanzaname:
    keyword1 = value1
    keyword2 = value2
    keyword3 = value3
     . . .
```

The *stanzaname*: must start at the first column within the file. A *stanzaname* specified in the **/etc/system** file is the name of the special file for the kernel device driver (prefixed with **/dev/**, that is, **/dev/***stanzaname*).

The *keyword* = *value* pairs must follow on consecutive lines with no blank lines between the *keyword* = *value* pair lines or between *stanzaname:* and the first *keyword* = *value* pair. Comments in these files require an * (asterisk) in the first column of every comment line. Stanzas must be separated by a minimum of two blank lines (that is, new-line characters in the first column).

/etc/system File

The **/etc/system** file contains primarily two types of stanzas. One type deals with minidisks (partitions on the fixed disks) and is not important to this discussion. The second type is the device stanza. This second type is required by users interested in installing their own VRM kernel device drivers. See "system" on page 4-139 for the file contents. The following is an example of a stanza for a printer.

```
lp0:
* IBM PC Graphics Printer (5152)
   name = 5152
   crname = true
   minor = c0
   vint = 4
   iodn = 12000
   kaf_file = /etc/ddi/pprinter.kaf
   kaf_use = kparallel
   file = /etc/ddi/pprinter
   noddi = false
   dtype = printer
* Printer
   switchable = true
* Coprocessor Device
   specproc = cfgaqcfg
   shared = false
   noduplicate = false
   dname = lp
   noshow = false
   aflag = true
* IBM Mono Disp & Paral Prntr
   adp = mp,sp1,sp2
   use = d5152mp
   nname = 5152mp
   driver = u5152mp
```

/etc/master File

The **/etc/master** file also contains two types of stanzas. The first type describes information about the AIX kernel device driver. The second type of stanza describes information about the VRM device driver. The following example shows both types which also have references in the **/etc/system** file previously shown.

```
* printer drivers

 u5152mp:                      <==== Kernel stanza
 major = 6
 prefix = lp
 routines = open,close,write,ioctl,init
 maxminor = 8
 vdriver = v5152mp
 config = vrcmain


 v5152mp:
 iocn = 2011                          <==== VRM stanza
 code = /vrm/vrmdd/vpptr
 ctype = vdrvr
```

/etc/ddi Directory

The files in the **/etc/ddi** directory contain information specific to individual device types (such as, **/etc/ddi/pprinter** concerns parallel attached printers). In general, an **/etc/ddi** file exists for each type of peripheral device. The **/etc/ddi/pprinter** file contains information about both the parallel printer devices as well as a description of the hardware adapter cards to which the printer must be connected. The **ddi** file must contain certain hardware adapter information that is required to define a VRM device driver. It may contain information that allows the **devices** command to operate on the particular device. (This is true for IBM devices). Other information is strictly dependent on the device requirements.

Customize File Relationships

As previously stated, the **/etc/system** file is the focal point for all devices currently configured into a particular RT PC system. The **/etc/system** stanzas are keyword pointers to stanzas in other customize files. The information contained in a particular **/etc/system** stanza in addition to the information in the stanzas indicated (target) completely describes that device. The relational keywords in the stanzas are pointers to other files and other stanzas. Pointers in the /etc/system file are:

```
system_stanza:
driver = driver_stanza
use = ddi_stanza_name
```

```
file = ddi_file_name
kaf_file = kaf_file_name
kaf_use = kaf_stanza_name
vdmgr = device1,device2, . . .
 . . .
```

The following describes the relational keywords in the **/etc/system** file:

driver Specifies the stanza name of the AIX kernel device driver stanza in the **/etc/master** file. In certain instances, there is no direct kernel device driver and the **driver** = *keyword* points directly to the VRM device driver stanza in the **/etc/master** file.

file Specifies the device dependent information (ddi) filename (full pathname of file).

use Specifies the stanza name in the ddi file that contains the device and adapter specific information.

kaf_file Specifies the name of the attribute file (used by the **devices** command and the IBM-supplied customize helper program) that contains instructions as to how the device information is to be processed.

kaf_use Specifies the stanza name in the kaf_file.

vdmgr Used only in special cases to describe a VRM device manager (see *Virtual Resource Manager Technical Reference* for information concerning device managers). The value list (separated by a **,** (comma) (no blanks are allowed) are the stanza names (in this file) of the devices managed by this device manager.

Pointers in the **/etc/master** file are:

```
kernel_master_stanza:
vdriver = driver_stanza
 . . .
```

```
VRM_master_stanza:

    code = /vrm/vrmdd/driver
     . . .
```

The following describes the relational keywords in the **/etc/master** file.

vdriver Specifies the name of the VRM device driver stanza within this same file.

code Specifies the full path name to the executable VRM device driver code.

Writing a Customize Helper Program

The customize helper program is responsible to locate, convert, and construct the **Define_Device** structure unique to the VRM device driver being loaded in the character string it receives. The helper program must additionally locate, convert, and construct the AIX kernel device driver structure that calls the AIX kernel device driver initialization routine.

Both program structures are declared in header file **/usr/include/sys/kcfg.h**, which must be included in the source code of the **customize helper** program. After both program structures are constructed, the **customize helper** program issues **ioctl** system calls to the **/dev/config** kernel device driver to complete the function.

A customize helper program is required to issue the **Define_Device** SVC for each VRM device driver. It is also required to call the initialization routine of the AIX kernel device driver associated with this VRM device driver. Use the **cfgabdds** subroutine to build the **Define_Device** structure and issue the SVC. See "cfgabdds" on page 3-13 for information about this subroutine.

Developing Device Drivers Before the Customize Helper Program

Sometimes the VRM or AIX kernel device drivers are ready to be tested before the **customize helper** program is available and the device driver structures are finalized. At that time a mechanism to allow testing to continue without the existence of the final design is needed. This requires the ability to temporarily circumvent the need for a customize helper program. The **ddi** and **uinfo** keywords can be used to create this temporary mechanism for both VRM and AIX kernel device drivers. The **ddi** keyword can be used to create temporary VRM DDS structures for device drivers under development, and the **uinfo** keyword can be used to initialize AIX kernel device drivers.

Warning: If either the **ddi** or **uinfo** keyword is used, then the **config** keyword, which specifies the customize helper program, cannot be used. If these keywords appear in the same device configuration stanza, or in stanzas that are referenced for a single device, then undefined results will occur.

Both the **ddi** and **uinfo** keywords map ASCII character string values as data for the corresponding driver program structures. Both require that the number of ASCII character value fields be a multiple of words in the RT PC environment. That is, no partial word of the structure data may be used. A word in the RT PC environment is 4 bytes. Therefore, to represent one word in a device driver structure requires exactly eight ASCII characters in the **ddi** value field. For example, to put the value 0xfedcba98 (note: exactly one word) in a structure requires only that the string `ddi = fedcba98` be included in the device configuration stanza. To initialize the corresponding kernel driver in the same manner would require that the string `uinfo = fedcba98` be included in the device configuration stanza. The data (such as converted values for these keywords) is included in the proper program structures in particular fields. The user of this mechanism is responsible to manually calculate structure **offsets** and **length** fields and to include this information in the character strings.

The **ddi** keyword data starts at the **union** in the **defdev** structure, which is defined in **<sys/kcfg.h>**. This structure makes it clear that the user must calculate three offset fields as well as three length fields in order to complete it. The **vrmconfig** command adds data in the **defdev** structure preceding the **union** declaration.

The **uinfo** keyword adds data in the program structure starting at the **union** in the **unxdrv** structure, which is defined in **<sys/kcfg.h>**. Again, the data must be an exact multiple of words (eight ASCII characters per word). The **vrmconfig** command adds data to the program structure preceding the **union** declaration.

Rebuilding the AIX Kernel

If the you have written an AIX kernel device driver, then you must rebuild the AIX kernel to include the new kernel driver. You may also rebuild the kernel at other times in order to change system parameters in the **sysparms** stanza of the **/etc/master** file. The procedure to rebuild the kernel is essentially the same whether you are installing a device driver in the AIX kernel or not. The following procedure applies to any rebuild operation, except for steps 1, 2, and 3, which apply only when installing a new device driver.

Use the following steps to rebuild the kernel:

1. After you have written the new kernel device driver, compile it:

   ```
   cc  -c  driver_name.c
   ```

2. Archive the new kernel device driver into the **/usr/sys/lib2** library, which contains the kernel device drivers:

   ```
   ar  r  /usr/sys/lib2  driver_name.o
   ```

3. Modify the **/etc/system** and **/etc/master** files to reflect the new device driver information.

   - Add new stanzas to **/etc/system** and **/etc/master**. A device named **/dev/***device* should have a stanza in the **/etc/system** file named *device.*

   - Select keywords and values for the new stanza that do not conflict with the existing machine environment. For example, the major device number defined by **major** = *num* in the **/etc/master** stanza must not conflict with that of an existing stanza.

   See "system" on page 4-139 and "master" on page 4-98 for more information about constructing stanzas.

4. If necessary, modify other parameters in **/etc/master**, such as the **sysparms** stanza.

5. Change the current directory to **/usr/sys** and run the **make** command:

   ```
   cd  /usr/sys
   make
   ```

   The **make** command produces a file named **/usr/sys/unix.std**, which contains the newly built AIX kernel, and a shell procedure named **/usr/sys/specials**.

6. Run the shell procedure created by the **make** command:

   ```
   /usr/sys/specials
   ```

7. Rename the existing AIX kernel as a precaution until the new kernel has been successfully tested:

   ```
   mv  /unix.std  /unix.old
   ```

8. Move the new AIX kernel to the root directory and create another link to it named **/unix**:

   ```
   mv  /usr/sys/unix.std  /unix.std
   ln  /unix.std  /unix
   ```

9. Shut the system down:

   ```
   shutdown
   ```

10. Restart the system by pressing the **Ctrl-Alt-Pause** key sequence. This loads and runs the new AIX kernel with the modified customization files.

11. When you are satisfied with performance of the new kernel, delete the old kernel (`/unix.old`).

Appendix D. Porting DOS 3.0 Applications

This section is intended for an experienced DOS 3.0 programmer that desires to port applications to the RT PC system using the AIX operating system. This section directs you to the proper section of RT PC documentation that contains detailed information required to port an application.

High-Level Languages

The RT PC has several high-level languages that are designed to allow application migration and portability or both with minimal changes to the application. The languages provided are Basic, Pascal, Fortran, and C. The Basic and Pascal languages contain the same syntax and semantics as the PC DOS versions of these languages, in addition to extensions to the language in order to take advantage of the RT PC architectural features. For a complete description of the languages, their extensions, and any differences between them and the PC DOS versions of the language, see *BASIC Language Reference* and *Pascal Compiler Language Reference.*

DOS File System

The DOS file system and the RT PC native file systems are different. However, a complete set of commands and run-time subroutines are provided for the application developer to mask these file system differences. See the **dosdel, dosdir, dosread, doswrite** commands in *AIX Operating System Commands Reference*, for a complete description of the commands that manipulate a DOS file system. The RT PC system provides a complete set of run-time subroutines that allow an application to deal with both DOS and native RT PC file systems. See "DOS services library," for a description of the DOS library and the DOS RT PC subroutine calls and see Chapter 3, "Subroutines," for a complete description of each file system subroutine.

Note: These routines are intended to handle ASCII files in a transparent fashion. The application must handle binary data within a file.

DOS Function Calls

Most DOS function calls map to equivalent AIX operating system functions in a very straightforward way. The following is a detailed listing of DOS function calls and the equivalent AIX operating system functions that provide the same services.

Table of DOS 3.0 Function Calls

Function Calls	AIX Equivalent Function	Notes
00H Program terminate	exit	See "hft" on page 6-23.
01H Keyboard input	KSR echo, getc	
02H Display output	putc	
03H Auxiliary input	read	See "tty" on page 6-131.
04H Auxiliary output	write	See "tty" on page 6-131.
05H Printer output	write	See "lp" on page 6-98.
06H Direct console I/O	getc, putc	
07H Direct console input without echo	getchar	
08H Console input without echo	getchar	
09H Print string	puts	
0AH Buffered keyboard input	gets	
0BH Check standard input status	ioctl	See "hft" on page 6-23.
0CH Clear keyboard buffer, invoke a keyboard function	ioctl	See "hft" on page 6-23.
0DH Disk reset	sync	
0EH Select disk	Set environment variable	See "DOS services library" on page 3-65.

Function Calls	AIX Equivalent Function	Notes
0FH Open file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
10H Close file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
11H Search for first entry	mapped	All FCB functions should be mapped to the equivalent file handling functions.
12H Search for next entry	mapped	All FCB functions should be mapped to the equivalent file handling functions.
13H Delete file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
14H Sequential read	mapped	All FCB functions should be mapped to the equivalent file handling functions.
15H Sequential write	mapped	All FCB functions should be mapped to the equivalent file handling functions.
16H Create file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
17H Rename file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
18H Used internally by DOS	DOS	
19H Current disk	dospwd	
1AH Set disk transfer address	N/A	
1BH Allocation table information	dosstat	
1CH Allocation table information for specific device	dosustat	

Function Calls	AIX Equivalent Function	Notes
1DH Used internally by DOS	N/A	
1EH Used internally by DOS	N/A	
1FH Used internally by DOS	N/A	
20H Used internally by DOS	N/A	
21H Random read	mapped	All FCB functions should be mapped to the equivalent file handling functions.
22H Random write	mapped	All FCB functions should be mapped to the equivalent file handling functions.
23H File size	mapped	All FCB functions should be mapped to the equivalent file handling functions.
24H Set relative record field	mapped	All FCB functions should be mapped to the equivalent file handling functions.
25H Set interrupt vector	N/A	Part of system configuration.
26H Create new program segment	exec, dosexecvec	
27H Random block read	mapped	All FCB functions should be mapped to the equivalent file handling functions.
28H Random block write	mapped	All FCB functions should be mapped to the equivalent file handling functions.
29H Parse filename	N/A	Not useful for path names.
2AH Get date	time	
2BH Set date	stime	Must have superuser authority.
2CH Get time	time	Must have superuser authority.
2DH Set time	stime	
2EH Set/reset verify switch	ioctl	See “hd” on page 6-20.

Function Calls	AIX Equivalent Function	Notes
2FH Get disk transfer address	N/A	
30H Get DOS version number	uname	
31H Terminate process and remain resident	N/A	See "fork" on page 2-46, "exec: execl, execv, execle, execve, execlp, execvp" on page 2-34, and "signal" on page 2-145.
32H Used internally by DOS	N/A	
33H Ctrl-Break check	N/A	Break key generates signal to process.
34H Used internally by DOS	N/A	
35H Get vector	N/A	
36H Get disk free space	ustat, dosustatl	
37H Used internally by DOS	N/A	
38H Set or get country dependent information	N/A	
39H Create subdirectory (MKDIR)	mkdir, dosmkdir	
3AH Remove subdirectory (RMDIR)	rm, dosrmdir	
3BH Change current directory (CHDIR)	chdir, doschdir	
3CH Create a file (CREAT)	creat, doscreate	
3DH Open a file	open, dosopen	
3EH Close a file handle	close, dosclose	
3FH Read from a file or device	read, dosread	
40H Write to a file or device	write, doswrite	
41H Delete a file from a specified directory (UNLINK)	unlink, dosunlink	
42H Move file read/write pointer (LSEEK)	lseek, doslseek	

Function Calls	AIX Equivalent Function	Notes
43H Change file mode (CHMOD)	chmod, doschmod	
44H I/O control for devices (IOCTL)	ioctl	
45H Duplicate a file handle (DUP)	dup, dosdup	
46H Force a duplicate of a file handle (FORCDUP)	fcntl	
47H Get current directory	getcwd, dospwd	
48H Allocate memory	malloc	
49H Free allocated memory	free	
4AH Modify allocated memory blocks (SETBLOCK)	realloc	
4BH Load or execute a program (EXEC)	exec, dosexecvec	
4CH Terminal a process (EXIT)	exit	
4DH Get return code of a sub-process (WAIT)	wait	
4EH Find first matching file (FIND FIRST)	dosfirst	
4FH Find next machine file	dosnext	
50H Used internally by DOS	N/A	
51H Used internally by DOS	N/A	
52H Used internally by DOS	N/A	
53H Used internally by DOS	N/A	
54H Get verify setting	ioctl	See “hd” on page 6-20.
55H Used internally by DOS	N/A	
56H Rename a file	dosrename	
57H Get/set a file's date and time	utimec, dostatc, dostouch	

Function Calls	AIX Equivalent Function	Notes
58H Used internally by DOS	N/A	
59H Get extended error	errno, doserrno	
5AH Create temporary file	mktemp, dosmktemp	
5BH Create new file	N/A	
5CH Lock/unlock file access	lockf	
5DH Used internally by DOS	N/A	
5EH Used internally by DOS	N/A	
5FH Used internally by DOS	N/A	
60H Used internally by DOS	N/A	
61H Used internally by DOS	N/A	
62H Get PSP address	N/A	

RT PC Hardware Access

Normal access to the RT PC system hardware devices is by means of special files (see chapter 6 in this book). However, it is possible for the applications to gain direct control of the hardware display to perform high function graphic applications. To use the display in this fashion, see "Monitor Mode (MOM)" on page 6-73. An application can also gain direct access to the hardware I/O or to manipulate a device that is unknown to the system (see "bus" on page 6-5 and the *Virtual Resource Manager Technical Reference*).

For a complete description of the hardware and I/O devices supported by RT PC, see *Hardware Technical Reference.*

Differences Between IBM Personal Computer AT Processor and 032 Microprocessor

There are hardware differences between the processors that affect application data. For example, IBM Personal Computer AT integers are 2 bytes long and are typically stored in a left-reversed fashion on media; while 032 Microprocessor integers are 4 bytes long and are stored in sequence on the media. There are also differences in the way floating-point numbers are stored. The person developing applications is responsible for understanding and handling any differences between binary data that is exchanged between IBM Personal Computer AT architecture machines and 032 Microprocessor architecture machines. For a more detailed description of the 032 Microprocessor architecture, see *Assembler Language Reference* and *Personal Computer AT Coprocessor Services Technical Reference.*

Appendix E. Component Cross Reference

The following subroutines and subroutine libraries are packaged with the Extended Services Program:

Subroutine or Library	Page
dbminit	3-63
delete	3-63
fetch	3-63
firstkey	3-63
libdbm.a	3-63
libprint.a	4-115
nextkey	3-63
store	3-63

Figure E-1. Extended Services Subroutines

The following subroutines and subroutine libraries are packaged with the Multi-User Services Program:

Subroutine or Library	Page
arc	4-115
circle	4-115
closepl	4-115
cont	4-115
erase	4-115
label	4-115
libgsl.a	7-1

Figure E-2 (Part 1 of 2). Multi-User Services Subroutines

Subroutine or Library	Page
libplot.a	4-115
lib300.a	4-115
lib300s.a	4-115
lib300S.a	4-115
lib4014.a	4-115
lib450.a	4-115
line	4-115
linemod	4-115
move	4-115
openpl	4-115
point	4-115
space	4-115

Figure E-2 (Part 2 of 2). Multi-User Services Subroutines

All other system calls, subroutines, and subroutine libraries discussed in this book are packaged with the IBM RT PC AIX Operating System Licensed Program.

Appendix F. Glossary

access. To obtain data from or put data in storage.

access permission. A group of designations that determine who can access a particular AIX file and how the user may access the file.

account. The log in directory and other information that give a user access to the system.

activity manager. A collection of system-supplied tasks allowing users to manage their activities. Provides the ability to list current activities (Activity List) and to begin, cancel, hide, and activate activities.

All Points Addressable (APA) display. A display that allows each pel to be individually addressed. An APA display allows for images to be displayed that are not made up of images predefined in character boxes. Contrast with *character display.*

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic. Pertaining to a set of letters a through z.

alphanumeric character. Consisting of letters, numbers and often other symbols, such as punctuation marks and mathematical symbols.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute. An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

application. A program or group of programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

application program. A program used to perform an application or part of an application.

argument. Numbers, letters, or words that change the way a command works.

ASCII. See *American National Standard Code for Information Interchange.*

attribute. A characteristic. For example, the attribute for a displayed field could be blinking.

auto carrier return. The system function that places carrier returns automatically within the text and on the display. This is accomplished by moving whole words that exceed the line end zone to the next line.

backend. The program that sends output to a particular device. There are two types of backends: friendly and unfriendly.

background process. (1) A process that does not require operator intervention that can be run by the computer while the work station is used to do other work. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command.

backup copy. A copy, usually of a file or group of files, that is kept in case the original file or files are unintentionally changed or destroyed.

backup diskette. A diskette containing information copied from a fixed disk or from another diskette. It is used in case the original information becomes unusable.

bad block. A portion of a disk that can never be used reliably.

base address. The beginning address for resolving symbolic references to locations in storage.

base name. The last element to the right of a full path name. A filename specified without its parent directories.

batch printing. Queueing one or more documents to print as a separate job. The operator can type or revise additional documents at the same time. This is a background process.

batch processing. A processing method in which a program or programs process records with little or no operator action. This is a background process. Contrast with *interactive processing.*

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. Either of the binary digits 0 or 1 used in computers to store information. See also *byte.*

block. (1) A group of records that is recorded or processed as a unit. Same as *physical record.* (2) In data communications, a group of records that is recorded, processed, or sent as a unit. (3) A block is 512 bytes long. (4) A logical block is 2048 bytes long.

block file. A file listing the usage of blocks on a disk.

block special file. A special file that provides access to an input or output device is capable of supporting a file system. See also *character special file.*

bootstrap. A small program that loads larger programs during system initialization.

branch. In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met.

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

burst pages. On continuous-form paper, pages of output that can be separated at the perforations.

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. (1) To activate a program or procedure at its entry point. Compare with *load.*

callouts. An AIX kernel parameter establishing the maximum number of scheduled activities that can be pending simultaneously.

cancel. To end a task before it is completed.

carrier return. (1) In text data, the action causing line ending formatting to be performed at the current cursor location followed by a line advance of the cursor. Equivalent to the carriage return of a typewriter. (2) A keystroke generally indicating the end of a command line.

case sensitive. Able to distinguish between uppercase and lowercase letters.

character. A letter, digit, or other symbol.

character display. A display that uses a character generator to display predefined character boxes of images (characters) on the screen. This kind of display cannot address the screen any less than one character box at a time. Contrast with *All Points Addressable display.*

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with *function keys.*

character position. On a display, each location that a character or symbol can occupy.

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

character special file. A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. See also *block special file.*

character string. A sequence of consecutive characters.

character variable. The name of a character data item whose value may be assigned or changed while the program is running.

child. (1) Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. (2) In the AIX Operating System, child is a *process* spawned by a parent process that shares resources of parent process. Contrast with *parent.*

C language. A general-purpose programming language that is the primary language of the AIX Operating System.

class. Pertaining to the I/O characteristics of a device. AIX devices are classified as block or character.

close. (1) To end an activity and remove that window from the display.

code. (1) Instructions for the computer. (2) To write instructions for the computer; to *program.* (3) A representation of a condition, such as an error code.

code segment. See *segment.*

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

color display. A display device capable of displaying more than two colors and the shades produced via the two colors, as opposed to a monochrome display.

column. A vertical arrangement of text or numbers.

column headings. Text appearing near the top of columns of data for the purpose of identifying or titling.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command interpreter. A program that sends instructions to the kernel; also called an interface.

command line. The area of the screen where commands are displayed as they are typed.

command line editing keys. Keys for editing the command line.

command programming language. Facility that allows programming by the combination of commands rather than by writing statements in a conventional programming language.

compile. (1) To translate a program written in a high-level programming language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compress. (1) To move files and libraries together on disk to create one continuous area of unused space. (2) In data communications, to delete a series of duplicate characters in a character string.

concatenate. (1) To link together. (2) To join two character strings.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running.

configuration. The group of machines, devices, and programs that make up a computer system. See also *system customization.*

configuration file. A file that specifies the characteristics of a system or subsystem, for example, the AIX queueing system.

consistent. Pertaining to a file system, without internal discrepancies.

console. (1) The main AIX display station. (2) A device name associated with the main AIX display station.

constant. A data item with a value that does not change. Contrast with *variable.*

context search. A search through a file whose target is a character string.

control block. A storage area used by a program to hold control information.

control commands. Commands that allow conditional or looping logic flow in DOS Services procedures.

control program. Part of the AIX Operating System system that determines the order in which basic functions should be performed.

controlled cancel. The system action that ends the job step being run, and saves any new data already created. The job that is running can continue with the next job step.

copy. The action by which the user makes a whole or partial duplicate of already existing data.

crash. An unexpected interruption of computer service, usually due to a serious hardware or software malfunction.

current directory. The directory that is active, and can be displayed with the **pwd** command.

current line. The line on which the cursor is located.

current working directory. See *current directory.*

cursor. (1) A movable symbol (such as an underline) on a display, used to indicate to the operator where the next typed character will be placed or where the next action will be directed. (2) A marker that indicates the current data access location within a file.

cursor movement keys. The directional keys used to move the cursor.

customize. To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

cylinder. All fixed disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

daemon. See *daemon process.*

daemon process. A process begun by the root or the root shell that can be stopped only by the root. Daemon processes generally provide services that must be available at all times such as sending data to a printer.

data block. See *block.*

data communications. The transmission of data between computers, or remote devices or both (usually over long distance).

data stream. All information (data and control information) transmitted over a data link.

debug. (1) To detect, locate, and correct mistakes in a program. (2) To find the cause of problems detected in software.

default. A value that is used when no alternative is specified by the operator.

default directory. The directory name supplied by the operating system if none is specified.

default drive. The drive name supplied by the operating system if none is specified.

default value. A value stored in the system that is used when no other value is specified.

delete. To remove. For example, to delete a file.

dependent work station. A work station having little or no standalone capability, that must be connected to a host or server in order to provide any meaningful capability to the user.

device. An electrical or electronic machine that is designed for a specific purpose and that attaches to your computer, for example, a printer, plotter, disk drive, and so forth.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device name. A name reserved by the system that refers to a specific device.

diagnostic. Pertaining to the detection and isolation of an error.

diagnostic aid. A tool (procedure, program, reference manual) used to detect and isolate a device or program malfunction or error.

diagnostic routine. A computer program that recognizes, locates, and explains either a fault in equipment or a mistake in a computer program.

digit. Any of the numerals from 0 through 9.

directory. A type of file containing the names and controlling information for other files or other directories.

disable. To make nonfunctional.

discipline. Pertaining to the order in which requests are serviced, for example, first-come-first-served (fcfs) or shortest job next (sjn).

disk I/O. Fixed-disk input and output.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copies from the disk or another diskette.

diskette drive. The mechanism used to read and write information on diskettes.

display device. An output unit that gives a visual representation of data.

display screen. The part of the display device that displays information visually.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or received from the computer.

dump. (1) To copy the contents of all or part of storage, usually to an output device.
(2) Data that has been dumped.

dump diskette. A diskette that contains a dump or is prepared to receive a dump.

dump formatter. Program for analyzing a dump.

EBCDIC. See *extended binary-coded decimal interchange code.*

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. To modify the form or format of data.

edit buffer. A temporary storage area used by an editor.

editor. A program used to enter and modify programs, text, and other types of documents and data.

emulation. Imitation; for example, when one computer imitates the characteristics of another computer.

enable. To make functional.

enter. To send information to the computer by pressing the **Enter** key.

entry. A single input operation on a work station.

environment. The settings for shell variables and paths set associated with each process. These variables can be modified later by the user.

error-correct backspace. An editing key that performs editing based on a cursor position; the cursor is moved one position toward the beginning of the line, the character at the new cursor location is deleted, and all characters following the cursor are moved one position toward the beginning of the line (to fill the vacancy left by the deleted element).

escape character. A character that suppresses the special meaning of one or more characters that follow.

exit value. A numeric value that a command returns to indicate whether it completed successfully. Some commands return exit values that give other information, such as whether a file exists. Shell programs can test exit values to control branching and looping.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

feature. A programming or hardware option, usually available at an extra cost.

field. (1) An area in a record or panel used to contain a particular category of data. (2) The smallest component of a record that can be referred to by a name.

FIFO. See *first-in-first-out.*

file. A collection of related data that is stored and retrieved by an assigned name.

file name. The name used by a program to identify a file. See also *label.*

filename. In DOS, that portion of the file name that precedes the extension.

file specification (filespec). The name and location of a file. A file specification consists of a drive specifier, a path name, and a file name.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

filetab. An AIX kernel parameter establishing the maximum number of files that can be open simultaneously.

filter. A command that reads standard input data, modifies the data, and sends it to standard output.

first-in-first-out (FIFO). A named permanent pipe. A FIFO allows two unrelated processes to exchange information using a pipe connection.

fixed disk. A flat, circular, nonremoveable plate with a magnetizable surface layer on which data can be stored by magnetic recording.

fixed-disk drive. The mechanism used to read and write information on fixed disk.

flag. A modifier that appears on a command line with the command name that defines the action of the command. Flags in the AIX Operating System almost always are preceded by a dash.

font. A family or assortment of characters of a given size and style.

foreground. A mode of program execution in which the shell waits for the program specified on the command line to complete before returning your prompt.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

formatted diskette. A diskette on which control information for a particular computer system has been written but which may or may not contain any data.

free list. A list of available space on each file system. This is sometimes called the free-block list.

free-block list. See *free list.*

full path name. The name of any directory or file expressed as a string of directories and files beginning with the root directory.

function. A synonym for procedure. The C language treats a function as a data type that contains executable code and returns a single value to the calling function.

function keys. Keys that request actions but do not display or print characters. Included are the keys that normally produce a printed character, but when used with the code key produce a function instead. Compare with *character key.*

generation. For some remote systems, the translation of configuration information into machine language.

Gid. See *group number.*

global. Pertains to information available to more than one program or subroutine.

global action. An action having general applicability, independent of the context established by any task.

global character. The special characters * and ? that can be used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position.

global search. The process of having the system look through a document for specific characters, words, or groups of characters.

global variable. A symbol defined in one program module, but used in other independently assembled program modules.

graphic character. A character that can be displayed or printed.

group name. A name that uniquely identifies a group of users to the system.

group number (Gid). A unique number assigned to a group of related users. The group number can often be substituted in commands that take a group name as an argument.

hardware. The equipment, as opposed to the programming, of a computer system.

header. Constant text that is formatted to be in the top margin of one or more pages.

header label. A special set of records on a diskette describing the contents of the diskette.

here document. Data contained within a DOS Services program or procedure (also called *inline input*).

highlight. To emphasize an area on the display by any of several methods, such as brightening the area or reversing the color of characters within the area.

history file. A file containing a log of system actions and operator responses.

hog factor. In system accounting, an analysis of how many times each command was run, how much processor time and memory it used, and how intensive that use was.

home directory. (1) A directory associated with an individual user. (2) The user's current directory on login or after issuing the **cd** command with no argument.

I/O. See *input/output.*

ID. Identification.

IF expressions. Expressions within a procedure, used to test for a condition.

indirect block. A block containing pointers to other blocks. Indirect blocks can be single-indirect, double-indirect, or triple-indirect.

informational message. A message providing information to the operator, that does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs. See *initialize.*

initialize. To set counters, switches, addresses, or contents of storage to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer routine.

inline input. See *here document.*

i-node. The internal structure for managing files in the system. I-nodes contain all of the information pertaining to the node, type, owner, and location of a file. A table of i-nodes is stored near the beginning of a file system.

i-number. A number specifying a particular i-node on a file system.

inodetab. An AIX kernel parameter that establishes a table in memory for storing copies of i-nodes for all active files.

input. Data to be processed.

input device. Physical devices used to provide data to a computer.

input file. A file opened by a program so that the program can read from that file.

input list. A list of variables to which values are assigned from input data.

input redirection. The specification of an input source other than the standard one.

input-output file. A file opened for input and output use.

input-output device number. A value assigned to a device driver by the guest operating system or to the virtual device by the virtual resource manager. This number uniquely identifies the device regardless of whether it is real or virtual.

input/output (I/O). Pertaining to either input, output, or both between a computer and a device.

interactive processing. A processing method in which each system user action causes response from the program or the system. Contrast with *batch processing.*

interface. A shared boundary between two or more entities. An interface might be a hardware component to link two devices together or it might be a portion of storage or registers accessed by two or more computer programs.

interleave factor. Specification of the ratio between contiguous physical blocks (on a fixed-disk) and logically contiguous blocks (as in a file).

interrupt. (1) To temporarily stop a process. (2) In data communications, to take an action at a receiving station that causes the sending station to end a transmission. (3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

IPL. See *initial program load.*

job. (1) A unit of work to be done by a system. (2) One or more related procedures or programs grouped into a procedure.

job queue. A list, on disk, of jobs waiting to be processed by the system.

justify. To print a document with even right and left margins.

kbuffers. An AIX kernel parameter establishing the number of buffers that can be used by the kernel.

K-byte. See *kilobyte.*

kernel. The memory-resident part of the AIX Operating System containing functions needed immediately and frequently. The kernel supervises the input and output, manages and controls the hardware, and schedules the user processes for execution.

kernel parameters. Variables that specify how the kernel allocates certain system resources.

key pad. A physical grouping of keys on a keyboard (for example, numeric key pad, and cursor key pad).

keyboard. An input device consisting of various keys allowing the user to input data, control cursor and pointer locations, and to control the dialog between the user and the display station

keylock feature. A security feature in which a lock and key can be used to restrict the use of the display station.

keyword. One of the predefined words of a programming language; a reserved word.

keyword argument. One type of variable assignment that can be made on the command line.

kill. An AIX Operating System command that stops a process.

kill character. The character that is used to delete a line of characters entered after the user's prompt.

kilobyte. 1024 bytes.

kprocs. An AIX kernel parameter establishing the maximum number of processes that the kernel can run simultaneously.

label. (1) The name in the disk or diskette volume table of contents that identifies a file. See also *file name.* (2) The field of an instruction that assigns a symbolic name to the location at which the instruction begins, or such a symbolic name.

left margin. The area on a page between the left paper edge and the leftmost character position on the page.

left-adjust. The process of aligning lines of text at the left margin or at a tab setting such that the leftmost character in the line or filed is in the leftmost position. Contrast with *right-adjust.*

library. A collection of functions, calls, subroutines, or other data.

licensed program product (LPP). Software programs that remain the property of the manufacturer, for which customers pay a license fee.

line editor. An editor that modifies the contents of a file one line at a time.

linefeed. An ASCII character that causes an output device to move forward one line.

link. A connection between an i-node and one or more file names associated with it.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive, or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

loader. A program that reads run files into main storage, thus preparing them for execution.

local. Pertaining to a device directly connected to your system without the use of a communications line. Contrast with *remote.*

log. To record; for example, to log all messages on the system printer. A list of this type is called a log, such as an error log.

log in. To begin a session at a display station.

log in shell. The program, or command interpreter, started for a user at log in.

log off. To end a session at a display station.

log out. To end a session at a display station.

logical device. A file for conducting input or output with a physical device.

loop. A sequence of instructions performed repeatedly until an ending condition is reached.

main storage. The part of the processing unit where programs are run.

maintenance system. A special version of the AIX Operating System which is loaded from diskette and used to perform system management tasks.

major device number. A system identification number for each device or type of device.

mapped files. Files on the fixed-disk that are accessed as if they are in memory.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

matrix. An array arranged in rows and columns.

maxprocs. An AIX kernel parameter establishing the maximum number of processes that can be run simultaneously by a user.

memory. Storage on electronic chips. Examples of memory are random access memory, read only memory, or registers. See *storage.*

menu. A displayed list of items from which an operator can make a selection.

message. (1) A response from the system to inform the operator of a condition which may affect further processing of a current program. (2) Information sent from one user in a multi-user operating system to another.

minidisk. A logical division of a fixed disk.

minor device number. A number used to specify various types of information about a particular device, for example, to distinguish among several printers of the same type.

mode word. An i-node field that describes the type and state of the i-node.

modem. See *modulator-demodulator.*

modulation. Changing the frequency or size of one signal by using the frequency or size of another signal.

modulator-demodulator (modem). A device that converts data from the computer to a signal that can be transmitted on a communications line, and converts the signal received to data for the computer.

module. (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. (2) See *load module*.

mount. To make a file system accessible.

mountab. An AIX kernel parameter establishing the maximum number of file systems that can be mounted simultaneously.

multiprogramming. The processing of two or more programs at the same time.

multivolume file. A diskette file occupying more than one diskette.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

network. A collection of products connected by communication lines for information exchange between locations.

new-line character. A control character that causes the print or display position to move to the first position on the next line.

null. Having no value, containing nothing.

null character (NUL). The character hex 00, used to represent the absence of a printed or displayed character.

numeric. Pertaining to any of the digits 0 through 9.

object code. Machine-executable instruction, usually generated by a compiler from source code written in a higher level language. consists of directly executable machine code. For programs that must be linked, object code consists of relocatable machine code.

octal. A base eight numbering system.

open. (1) To make a file available to a program for processing.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

operation. A specific action (such as move, add, multiply, load) that the computer performs when requested.

operator. A symbol representing an operation to be done.

output. The result of processing data.

output devices. Physical devices used by a computer to present data to a user.

output file. A file that is opened by a program so that the program can write to that file.

output redirection. The specification of an output destination other than the standard one.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write output into a storage or file space that is already occupied by data.

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. A block of instructions, data, or both.

page space minidisk. The area on a fixed disk that temporarily stores instructions or data currently being run. See also *minidisk*.

pagination. The process of adjusting text to fit within margins and/or page boundaries.

paging. The action of transferring instructions, data, or both between real storage and external page storage.

parallel processing. The condition in which multiple tasks are being performed simultaneously within the same activity.

parameter. Information that the user supplies to a panel, command, or function.

parent. Pertaining to a secured resource, either a file or library, whose user list is shared with one or more other files or libraries. Contrast with *child.*

parent directory. The directory one level above the current directory.

partition. See *minidisk.*

password. A string of characters that, when entered along with a user identification, allows an operator to sign on to the system.

password security. A program product option that helps prevent the unauthorized use of a display station, by checking the password entered by each operator at sign-on.

path name. See *full path name* and *relative path name.*

pattern-matching character. Special characters such as * or ? that can be used in search patterns. Some used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position. Pattern-matching characters are also called wildcards.

permission code. A three-digit octal code, or a nine-letter alphabetic code, indicating the access permissions. The access permissions are read, write, and execute.

permission field. One of the three-character fields within the permissions column of a directory listing indicating the read, write, and run permissions for the file or directory owner, group, and all others.

phase. One of several stages file system checking and repair performed by the **fsck** command.

physical device. See *device.*

physical file. An indexed file containing data for which one or more alternative indexes have been created.

physical record. (1) A group of records recorded or processed as a unit. Same as *block.* (2) A unit of data moved into or out of the computer.

PID. See *process ID.*

pipe. To direct the data so that the output from one process becomes the input to another process.

pipeline. A direct, one-way connection between two or more processes.

pitch. A unit of width of typewriter type, based on the number of times a letter can be set in a linear inch. For example, 10-pitch type has 10 characters per inch.

platen. The support mechanism for paper on a printer, commonly cylindrical, against which printing mechanisms strike to produce an impression.

pointer. A logical connection between physical blocks.

port. (1) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer. (2) An access point for data input to or data output from a computer system. See *connector.*

position. The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. A DOS Services facility for assigning values from the command line to variables in a program.

print queue. A file containing a list of the names of files waiting to be printed.

printout. Information from the computer produced by a printer.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

priority number. A number that establishes the relative priority of printer requests.

privileged user. The account with superuser authority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communications facilities, or user errors as the source of the problem.

problem determination procedure. A prescribed sequence of steps aimed at recovery from, or circumvention of, problem conditions.

procedure. See *shell procedure.*

process. (1) A sequence of actions required to produce a desired result. (2) An entity receiving a portion of the processor's time for executing a program. (3) An activity within the system begun by entering a command, running a shell program, or being started by another process.

process accounting. An analysis of the use each process makes of the processing unit, memory, and I/O resources.

process ID (PID). A unique number assigned to a process that is running.

profile. (1) A file containing customized settings for a system or user (2) Data describing the significant features of a user, program, or device.

program. A file containing a set of instructions conforming to a particular programming language syntax.

prompt. A displayed request for information or operator action.

propagation time. The time necessary for a signal to travel from one point on a communications line to another.

qdaemon. The daemon process that maintains a list of outstanding jobs and sends them to the specified device at the appropriate time.

queue. A line or list formed by items waiting to be processed.

queued message. A message from the system that is added to a list of messages stored in a file for viewing by the user at a later time. This is in contrast to a message that is sent directly to the screen for the user to see immediately.

quit. A key, command, or action that tells the system to return to a previous state or stop a process.

quote. To mask the special meaning of certain characters; to cause them to be taken literally.

random access. An access mode in which records can be read from, written to, or removed from a file in any order.

readonly. Pertaining to file system mounting, a condition that allows data to be read, but not modified.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again.

redirect. To divert data from a process to a file or device to which it would not normally go.

reference count. In an i-node, a record of the total number of directory entries that refer to the i-node.

relational expression. A logical statement describing the relationship (such as greater than or equal) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a relational condition or a relational expression.

relative address. An address specified relative to the address of a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

relative addressing. A means of addressing instructions and data areas by designating their locations relative to some symbol.

relative path name. The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

remote. Pertaining to a system or device that is connected to your system through a communications line. Contrast with *local.*

reserved character. A character or symbol that has a special (non-literal) meaning unless quoted.

reserved word. A word that is defined in a programming language for a special purpose, and that must not appear as a user-declared identifier.

reset. To return a device or circuit to a clear state.

restore. To return to an original value or image. For example, to restore a library from diskette.

right adjust. The process of aligning lines of text at the right margin or tab setting such that the rightmost character in the line or file is in the rightmost position.

right justify. See right align.

right margin. The area on a page between the last text character and the right upper edge.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with *left-adjust.*

root. Another name sometimes used for superuser.

root directory. The top level of a tree-structured directory system.

root file system. The basic AIX Operating System file system, which contains operating system files and onto which other file systems can be mounted. The root file system is the file system that contains the files that are run to start the system running.

routine. A set of statements in a program causing the system to perform an operation or a series of related operations.

run. To cause a program, utility, or other machine function to be performed.

run-time environment. A collection of subroutines and shell variables that provide commonly used functions and information for system components.

scratch file. A file, usually used as a work file, that exists until the program that uses it ends.

screen. See *display screen.*

scroll. To move information vertically or horizontally to bring into view information that is outside the display screen boundaries.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

security. The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

segment. A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

separator. A character used to separate parts of a command or file.

sequential access. An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

session records. In the accounting system, a record of time connected and line usage for connected display stations, produced from log in and log out records.

set flags. Flags that can be put into effect with the DOS Services set command.

shared printer. A printer that is used by more than one work station.

shell. See *shell program.*

shell procedure. A series of commands combined in a file that carry out a particular function when the file is run or when the file is specified as an argument to the sh command. Shell procedures are frequently called shell scripts.

shell program. A program that accepts and interprets commands for the operating system (there is an AIX shell program and a DOS shell program).

DOS Services prompt. The character string on the command line indicating the the system can accept a command (typically the $ character).

DOS Services script. See *DOS Services* procedure.

DOS Services variables. Facilities of the DOS Services program for assigning variable values to constant names.

size field. In an i-node, a field that indicates the size, in bytes, of the file associated with the i-node.

software. Programs.

sort. To rearrange some or all of a group of items based upon the contents or characteristics of those items.

source diskette. The diskette containing data to be copied, compared, restored, or backed up.

source program. A set of instructions written in a programming language, that must be translated to machine language compiled before the program can be run.

special character. A character other than an alphabetic or numeric character. For example; *, +, and % are special characters.

special file. Special files are used in the AIX system to provide an interface to input/output devices. There is at least one special file for each device connected to the computer. Contrast with *directory* and *file.* See also *block special file* and *character special file.*

spool files. Files used in the transmission of data among devices.

standalone DOS Services. A limited version of the DOS Services program used for system maintenance.

standalone work station. A work station that can be used to preform tasks independent of (without being connected to) other resources such as servers or host systems.

standard error. The place where many programs place error messages.

standard input. The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command.

stanza. A group of lines in a file that together have a common function. Stanzas are usually separated by blank lines, and each stanza has a name.

statement. An instruction in a program or procedure.

status. (1) The current condition or state of a program or device. For example, the status of a printer. (2) The condition of the hardware or software, usually represented in a status code.

storage. (1) The location of saved information. (2) In contrast to memory, the saving of information on physical devices such as disk or tape. See *memory*.

storage device. A device for storing and/or retrieving data.

string. A linear sequence of entities such as characters or physical elements. Examples of strings are alphabetic string, binary element string, bit string, character string, search string, and symbol string.

su. See *superuser*.

subdirectory. A directory contained within another directory in the file system hierarchy.

subprogram. A program invoked by another program, such as a subshell.

subroutine. (1) A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

subscript. An integer or variable whose value refers to a particular element in a table or an array.

subshell. An instance of the DOS Services program started from an existing DOS Services program.

substring. A part of a character string.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or synchronously with, a controlling system.

superblock. The most critical part of the file system containing information about every allocation or deallocation of a block in the file system.

superuser (su). The user who can operate without the restrictions designed to prevent data loss or damage to the system (User ID 0).

superuser authority. The unrestricted ability to access and modify any part of the operating system associated with the user who manages the system. The authority obtained when one logs in as **root**.

system. The computer and its associated devices and programs.

system call. A request by an active process for a service by the system kernel.

system customization. A process of specifying the devices, programs, and users for a particular data processing system.

system date. The date assigned by the system user during setup and maintained by the system.

system dump. A copy of memory from all active programs (and their associated data) whenever an error stops the system. Contrast with *task dump*.

system management. The tasks involved in maintaining the system in good working order and modifying the system to meet changing requirements.

system parameters. See *kernel parameters.*

system profile. A file containing the default values used in system operations.

system unit. The part of the system that contains the processing unit, the disk drives, and the diskette drives.

system user. A person who uses a computer system.

target diskette. The diskette to be used to receive data from a source diskette.

task. A basic unit of work to be performed. Examples are a user task, a server task, and a processor task.

task dump. A copy of memory from a program that failed (and its associated data). Contrast with *system dump.*

terminal. An input/output device containing a keyboard and either a display device or a printer. Terminals usually are connected to a computer and allow a person to interact with the computer.

text. A type of data consisting of a set of linguistic characters (for example, alphabet, numbers, and symbols) and formatting controls.

text application. A program defined for the purpose of processing text data (for example, memos, reports, and letters).

text editing program. See *editor* and *text application.*

texttab. A kernel parameter establishing the size of the text table, in memory, that contains one entry each active shared program text segment.

trace. To record data that provides a history of events occurring in the system.

trace table. A storage area into which a record of the performance of computer program instructions is stored.

track. A circular path on the surface of a fixed disk or diskette on which information is magnetically recorded and from which recorded information is read.

trap. An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

tree-structured directories. A method for connecting directories such that each directory is listed in another directory except for the root directory, which is at the top of the tree.

truncate. To shorten a field or statement to a specified length.

typematic key. A key that repeats its function multiple times when held down.

typestyle. Characters of a given size, style and design.

Uid. See *user number.*

update. An improvement for some part of the system.

user. The name associated with an account.

user account. See *account.*

user ID. See *user number.*

user name. A name that uniquely identifies a user to the system.

user number (Uid). (1) A unique number identifying an operator to the system. This string of characters limits the functions and information the operator is allowed to use. The Uid can often be substituted in commands that take a user's name as an argument.

user profile. A file containing a description of user characteristics and defaults (for example, printer assignment, formats, group ID) to be conveyed to the system while the user is signed on.

utility. A service; in programming, a program that performs a common service function.

valid. (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

value. (1) In Usability Services, information selected or typed into a pop-up. (2) A set of characters or a quantity associated with a parameter or name. (3) In programming, the contents of a storage location.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with *constant.*

verify. To confirm the correctness of something.

version. Information in addition to an object's name that identifies different modification levels of the same logical object.

virtual device. A device that appears to the user as a separate entity but is actually a shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

virtual machine. A functional simulation of a computer and its related devices.

virtual machine interface (VMI). A software interface between work stations and the operating system. The VMI shields operating system software from hardware changes and low-level interfaces and provides for concurrent execution of multiple virtual machines.

virtual resource manager (VRM). A set of programs that manage the hardware resources (main storage, disk storage, display stations, and printers) of the system so that these resources can be used independently of each other.

virtual resources. See *virtual resource manager.*

virtual storage. Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

virtual terminal. Any of several logical equivalents of a display station available at a single physical display station.

Volume ID (Vol ID). A series of characters recorded on the diskette used to identify the diskette to the user and to the system.

VRM. See *virtual resource manager.*

wildcard. See *pattern-matching characters.*

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

work file. A file used for temporary storage of data being processed.

work station. A device at which an individual may transmit information to, or receive information from, a computer for the purpose of performing a task, for example, a display station or printer. See *programmable work station* and *dependent work station.*

working directory. See *current directory.*

wrap around. Movement of the point of reference in a file from the end of one line to the beginning of the next, or from one end of a file to the other.

Index

The IBM RT PC
Family

Reader's Comment Form

IBM RT PC AIX Operating System Technical Reference SC23-0809-0

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact the authorized IBM RT PC dealer in your area.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

Please Do Not Staple

Tape

IBM

The IBM RT PC
Family

Reader's Comment Form

IBM RT PC AIX Operating System Technical Reference

SC23-0809-0

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact the authorized IBM RT PC dealer in your area.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

IBM RT PC AIX Operating System Technical Reference ______ **SC23-0809-0**

Book Title **Order No.**

Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name ______

Company name ______

Street address ______

City, State, ZIP ______

No postage necessary if mailed in the U.S.A.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

Please Do Not Staple

Tape