AIX Version 3 for RISC System/6000™

Files Reference

SC23-2200-00

First Edition (March 1990)

This edition of the AIX Files Reference for IBM RISC System/6000 applies to IBM AIX Version 3 for RISC System/6000 and the following Licensed Programs:

IBM AIX Network Management/6000
IBM AIX System Network Architecture Services/6000
IBM AIX 3278/79 Emulation/6000
IBM AIX 3270 Host Connection Program/6000
IBM AIX Personal Computer Simulator/6000

This edition applies to all subsequent releases of these products until otherwise indicated in new releases or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM does not warrant that the contents of this publication or the accompanying source code examples, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying source code examples are error-free.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements or changes to the products and the programs described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized 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.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. We acknowledge the following institutions for their role in its development: the Electrical Engineering and Computer Sciences Department at the Berkeley Campus.

Portion of the code and documentation described in this book were derived from code and documentation developed under the auspices of the Regents of the University of California and have been acquired and modified under the provisions that the following copyright notice and permission notice appear:

> Redistribution and use in source and binary forms are permitted provided that this notice is preserved and that due credit is given to the University of California at Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "as is" without express or implied warranty.
>
>

Trademarks and Acknowledgements

The following trademarks and acknowledgements apply to this information:

ADM is a trademark of Lear Siegler, Inc.

AIX is a trademark of International Business Machines Corporation.

AIX/RT is a trademark of International Business Machines Corporation.

AIXwindows is a trademark of International Business Machines Corporation.

BSC is a trademark of BusiSoft Corporation.

Connect is a trademark of INTERACTIVE Systems Corporation.

DEC is a trademark of Digital Equipment Corporation.

DEC VT100, VT220, VT320, and VT330 are trademarks of Digital Equipment Corporation.

Emacs is a trademark of Unipress Software, Inc.

GL is a trademark of Iris Graphics Library.

IBM is a registered trademark of International Business Machines Corporation.

IN/ix is a trademark of INTERACTIVE Systems Corporation.

Sun OS and NFS are trademarks of Sun Microsystems, Inc.

OSF and OSF/Motif are trademarks of Open Software Foundation, Inc.

PAL is a trademark of International Business Machines Corporation.

PC XT is a trademark of International Business Machines Corporation.

Personal Computer AT and AT are trademarks of International Business Machines Corporation.

POSIX is a trademark of the Institute of Electrical and Electronic Engineers (IEEE).

RISC System/6000 is a trademark of International Business Machines Corporation.

RT is a trademark of International Business Machines Corporation.

SNA 3270 is a trademark of International Business Machines Corporation.

Systems Application Architecture is a trademark of International Business Machines Corporation.

The Source is a service mark of Source Telecomputing Corp., a subsidiary of The Reader's Digest Assn., Inc.

UNIX was developed and licensed by AT&T and is a registered trademark of AT&T Corporation.

VAX is a trademark of Digital Equipment Corporation.

WYSE is a trademark of the WYSE Corporation.

WY–50 is a trademark of the WYSE Corporation.

X/OPEN is a trademark of X/OPEN Company Limited.

3270 Personal Computer AT is a trademark of International Business Machines Corporation.

3270 Personal Computer AT/G is a trademark of International Business Machines Corporation.

3270 Personal Computer AT/GX Personal Telephone Manager is a trademark of International Business Machines Corporation.

Note to Users

The term "network information service (NIS)" is now used to refer to the service formerly known as "Yellow Pages." The functionality remains the same; only the name has changed. The name "Yellow Pages" is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

Legal Notice to Users Issued by Sun Microsystems, Inc.

"Yellow Pages" is a registered trademark in the United Kingdom of British Telecommunications plc, and may also be a trademark of various telephone companies around the world. Sun will be revising future versions of software and documentation to remove references to "Yellow Pages."

About This Book

This book, *AIX Files Reference for IBM RISC System/6000*, describes the files used by the Advanced Interactive Executive Operating System (AIX) for use on the RISC System/6000. The various system files, file formats, special files, header files, and directories used by AIX, its subsystems, and certain optional program products are covered in the Files Reference.

Who Should Use This Book

This book is intended for system managers and experienced C programmers as well as end users with some knowledge of the AIX operating system. Readers of this book should have a basic understanding of the workings of the AIX operating system, its components, and any related subsystems for which file information is provided.

How to Use This Book

Overview of Contents

This book contains sections on the system files, special files, header files, and directories that are provided with the operating system and optional program products. File formats required for certain files that are generated by the system or an optional program are also presented in a section of this book.

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Related Publications

The following books contain information about or related to the AIX files:

- *IBM RISC System/6000 Quick Start Kit*, Order Number SC23–2195.
- *AIX General Concepts and Procedures for IBM RISC System/6000*, Order Number SC23–2202.
- *AIX Editing Concepts and Procedures for IBM RISC System/6000*, Order Number SC23–2212–0.

- *AIX Communication Concepts and Procedures for IBM RISC System/6000*, Order Number SC23–2203.
- *AIX Commands Reference for IBM RISC System/6000*, Order Number SC23–2199.
- *AIX Calls and Subroutines Reference for IBM RISC System/6000*, Order Number SC23–2198.
- *AIX General Programming Concepts for IBM RISC System/6000*, Order Number SC23–2205.
- *AIX User Interface Programming Concepts for IBM RISC System/6000*, Order Number SC23–2209.
- *AIX Communications Programming Concepts for IBM RISC System/6000*, Order Number SC23–2206.
- *AIX Graphics Programming Concepts for IBM RISC System/6000*, Order Number SC23–2208.
- *IBM 3270 Connection Technical Reference*, Order Number GA23–0339.

Ordering Additional Copies of This Book

To order additional copies of this book, use Order Number SC23–2200.

Contents

Chapter 1. AIX System Files

backup File

Purpose

Copies the file system onto temporary storage media.

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

Header Records

The different types of header records for the AIX Version 3 by name backups are:

FS_VOLUME Exists on every volume and holds the volume label.

FS_NAME_X Holds a description of a file backed up by name.

FS_END Indicates the end of the backup. This header appears at the end of the last volume.

The different types of header records for the AIX Version 3 by i-node and name backups are:

TS_TAPE Exists on every volume and holds the volume label.

TS_BITS Describes the directory structure.

TS_CLRI Describes the unused i-node numbers on the backup system.

TS_INODE Describes the file.

TS_ADDR A continuation of the preceding file.

TS_END Indicates the end of the backup.

The descriptions of the fields of the header structure for by i-node backups are:

c_type The header type.

c_date The current dump date.

c_ddate The file system dump date.

c_volume The volume number.

c_tapea The number of the current header record.

c_inumber The inode number on this record.

c_magic The magic number.

c_checksum The value that would make the record sum to the **CHECKSUM** value.

bsd_c_dinode A copy of the bsd inode as it appears on the bsd filesystem.

c_count The number of characters in the **c_addr** field.

c_addr — A character array that describes the blocks being dumped for the file.

xix_flag — Set to the **XIX_MAGIC** value if doing the backup of an AIX Version 3 file system.

xix_dinode — The real dinode from the AIX Version 3 file system.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a **TS_END** record and then the tapemark.

For more information on AIX Version 2 by name and by i-node header formats please consult your Version 2 documentation.

By Name Format

The format of an AIX Version 3 by name backup is:

FS_VOLUME
FS_NAME_X (before each file)
File Data
FS_END

The AIX Version 3 header formats for by name backups are not the same as the Version 2 header formats.

By i-node Format

The format of an AIX Version 3 by i-node backup follows:

TS_VOLUME
TS_BITS
TS_CLRI
TS_INODE
TS_END

A detailed description of the by i-node header file follows:

```
union u_spcl {
        char dummy[TP_BSIZE];
        struct s_spcl {
            int         c_type;                 /* 4 */
            time_t      c_date;                 /* 8 * /
            time_t      c_ddate;                /* 12 */
            int         c_volume;               /* 16 */
            daddr_t     c_tapea;                /* 20 */
            ino_t       c_inumber;              /* 24 */
            int         c_magic;                /* 28 */
            int         c_checksum;             /* 32 */
            struct bsd_dinode   bsd_c_dinode;   /* 160 */
            int         c_count;                /* 164 */
            char        c_addr[TP_NINDIR];      /* 676 */
            int         xix_flag;               /* 680 */
            struct dinode       xix_dinode;     /* 800 */
        } s_spcl;
} u_spcl;
```

Constants

Constants used to distinguish these different types of headers and define other variables are:

```
#define OSF_MAGIC          (int)60011
#define NFS_MAGIC          (int)60012  /* New File System Magic */
#define XIX_MAGIC          (int)60013  /* Magic number for AIXv3 */
#define BYNAME_MAGIC       (int)60011  /* 2.x magic number */2.x
#define PACKED_MAGIC       (int)60012  /* 2.x magic number for
                                       Huffman packed format*/
#define CHECKSUM           (int)84446  /* checksum magic number */
#define TP_BSIZE           1024        /* tape block size */
#define TP_NINDIR          (TP_BSIZE/2)/* number of indirect pointers
                                       in an inode record */
#define FS_VOLUME          0           /* denotes a volume header
#define FS_END  7                      /* denotes an end of backup */
#define FS_NAME_X          10          /* denotes file header */
#define SIZSTR  16                     /*string size in volume header
#define DUMNAME 4                      /* dummy name length for
                                       FS_NAME_X */
#define FXLEN              80          /* length of file index */
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Related Information

The **filesystems** file format.

The **backup** command, **pack** command, **restore** command.

The Backup Overview in *General Concepts and Procedures* provides information on different methods of backing up, restoring process, different types of backup media, and guidelines for backup policies.

The File Systems Overview in *General Concepts and Procedures* explains file system types, management, structure, and maintenance.

bincmds File

Purpose

Contains the shell commands that process audit bin data.

Description

The **/etc/security/audit/bincmds** file is an ASCII template file that contains the backend commands that process audit bin file records. The path name of this file is defined in the bin stanza of the **/etc/security/audit/config** file.

This file contains command lines each composed of one or more commands with input and output that can be piped together or redirected. Although the commands usually are one or more of the audit system commands (the **auditcat** command, the **auditpr** command, the **auditselect** command), this is not a requirement.

As each bin file is filled by the kernel, the **auditbin** daemon invokes each command to process the bin records, substituting the names of the current bin file and the audit trail file for any **$trail** and **$bin** strings in the commands.

The commands are executed by the trusted shell (TSH). This means that the path names in the commands must be absolute, and that environment variable substitution may be limited. See the discussion of the **tsh** command for more information.

Security

Access Control: This file should grant read (r) access to the root user and members of the audit group and grant write (w) access only to the root user.

Examples

1. To compress audit bin records and append them to the system audit trail file, include the following line in the **/etc/security/audit/bincmds** file:

   ```
   /etc/auditcat -p -o $trail $bin
   ```

   When the command runs, the names of the current bin file and the system audit trail file are substituted for the **$bin** and **$trail** strings. Records are compressed and appended to the **/audit/trail** file.

2. To select the audit events from each bin file that are unsuccessful because of authentication or privilege reasons, pack the audit events in a bin, and append the events to the **/audit/trail.violations** file, you must include the following line in the **/etc/security/audit/bincmds** file:

   ```
   /etc/auditselect -e "result == fail_auth || \
   result == fail_priv" $bin || \
   /etc/auditcat -p -o \
   /audit/trail.violations
   ```

3. To create a hard copy audit log of all local user authentication audit events, include the following line in the **/etc/security/audit/bincmds** file:

   ```
   /etc/auditselect -e "event == USER_Login || \
   event = USER_SU" $bin || \
   /etc/auditpr -t2 -v >/dev/lpr3
   ```

   Adjust the printer name to fit your requirements.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/audit/bincmds	Specified the path to the file.
/etc/security/audit/config	Contains audit system configuration information.
/etc/security/audit/events	Contains the audit events of the system.
/etc/security/audit/objects	Contains information about audited objects (files).
/etc/security/audit/streamcmds	Contains auditstream commands.

Related Information

The **audit** command, **auditbin** daemon, **auditcat** command, **auditpr** command, **auditselect** command, **tsh** command.

To see the steps you must take to establish an Auditing System, refer to How to Set Up an Auditing System in *General Concepts and Procedures*.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

config File

Purpose

Contains audit system configuration information.

Description

The **/etc/security/audit/config** file is an ASCII stanza file that contains audit system configuration information. This file contains five stanzas: start, bin, stream, classes, and users.

start Stanza

The start stanza contains the attributes used by the **audit start** command to initialize the audit system. The format follows:

```
start:
  binmode = off | on | panic
  streammode = off | on
```

The attributes are defined as follows:

binmode Controls whether bin collection is used.

- **off** Bin collection is not used. This is the default value.
- **on** Bin collection is used. This value starts the **auditbin** daemon.
- **panic** Bin collection is used. This value starts the **auditbin** daemon. If an audit record cannot be written to a bin, the kernel shuts down the operating system. This mode should be specified for conditions during which the system must be working properly.

streammode Controls whether stream data collection, as defined in the **/etc/security/streamcmds** file, is configured at the start up of the audit system.

- **off** Stream data collection is not enabled.
- **on** Stream data collection is enabled. This is the default value.

If neither collection mode is defined or if both modes are in the off state, subsystem configuration is done.

bin Stanza

The bin stanza contains the attributes used by the **auditbin** daemon to set up bin mode auditing. The format follows:

```
bin:
  trail = PathName
  bin1 = PathName
  bin2 = PathName
  binsize = DecimalString
  cmds = PathName
```

Bin mode parameters are defined as follows:

trail Specifies the path name of the audit trail file. When this is defined, the **auditbin** daemon can substitute the path name of the audit trail file for the **$trail** string in the backend commands that it calls.

bin1 Specifies the path name that the **auditbin** daemon uses for its primary bin file. If the **$bin** string is the parameter value, the **auditbin** daemon substitutes the name of the current bin file.

bin2 Specifies the path name that the **auditbin** daemon uses for its secondary bin file. If the **$bin** string is the parameter value, the **auditbin** daemon substitutes the name of the current bin file.

binsize Specifies a decimal integer string that defines the threshold size (in bytes) of each audit bin.

cmds Specifies the path name of the file that contains the audit backend commands called by the **auditbin** daemon. The file contains command lines, each composed of one or more backend commands with input and output that can be piped together or redirected. See the description of the **/etc/security/audit/bincmds** file for more information.

stream Stanza

The stream stanza contains the attributes that the **audit start** command uses to set up initial stream mode auditing. The format follows:

```
cmds = PathName
```

The *PathName* parameter identifies the file that contains the stream commands that are started at the initialization of the audit system. These commands can use shell piping and redirection, but no substitution of path names is performed on **$trail** or **$bin** strings.

classes Stanza

The classes stanza defines audit classes (sets of audit events) to the system.

Each audit class name must be less than 16 characters and be unique on the system. The system supports up to 32 audit classes, with ALL as the last class. The audit events in the class must be defined in the **/etc/security/audit/events** file.

```
classes:
        auditclass = auditevent, ...auditevent
```

See Examples for an illustration.

users Stanza

The users stanza defines audit classes (sets of events) for each user. The classes are defined to the operating system kernel.

The format is as follows:

```
users:
    UserName = auditclass, ... auditclass
```

Each *UserName* attribute must be the login name of a system user, and each *auditclass* parameter should be defined in the classes stanza.

To establish the audit activities for a user, use the **chuser** command with the **auditclasses** attribute.

Security

Access Control: This file should grant read (r) access to the root user and members of the audit group and write (w) access only to the root user.

Event	Information
AUD_CONFIG_WR	filename

Examples

1. To define audit classes, add a line to the `classes` stanza of the **/etc/security/audit/config** file for each set of events that you want to assign to a class:

   ```
   classes:
     general = USER_SU,PASSWORD_Change,FILE_Unlink,
       FILE_Link,FILE_Remove
     system = USER_Change,GROUP_Change,USER_Create,
       GROUP_Create
     init = login, USER_Logout
   ```

   These specific audit events and audit classes are described in How to Set Up an Audit System.

2. To establish the audit activities for each user, use the **chuser** command with the **auditclasses** attribute for each user for whom you want to define audit classes (sets of audit events):

   ```
   chuser "auditclasses=general,init,system" dave

   chuser "auditclasses=general,init" mary
   ```

   These **chuser** commands create the following lines in the users stanza of the **/etc/security/audit/config** file:

   ```
   users:
    dave=general,init,system
    mary=general,init
   ```

   This configuration includes `dave`, the administrator of the system, and `mary`, an employee who updates information.

3. To enable the auditing system, turn on bin data collection, and turn off initial stream data collection, add the following to the `start` stanza of the **/etc/security/audit/config** file:

   ```
   start:
     binmode = on
     streammode = off
   ```

4. To enable the **auditbin** daemon to set up bin collection, add attributes to the `bin` stanza of the **/etc/security/audit/config** file:

```
start:
  binmode = on
bin:
  trail = /audit/trail
  bin1 = /audit/bin1
  bin2 = /audit/bin2
  binsize = 25000
  cmds = /etc/security/audit/bincmds
```

   The attribute values in the preceding stanza enable the audit system to collect bin files of data and store the records in a long–term audit trail.

5. To enable the **auditbin** daemon to set up stream collection, add lines to the start stanza and the stream stanza of the **/etc/security/audit/config** file:

```
start:
    streammode = on

stream:
    cmds = /etc/security/audit/streamcmds
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/audit/config	Specifies the path to the file.
/etc/security/audit/objects	Contains information about audited objects.
/etc/security/audit/events	Contains the audit events of the system.
/etc/security/audit/bincmds	Contains auditbin backend commands.
/etc/security/audit/streamcmds	Contains auditstream commands.

Related Information

The **audit** command, **auditbin** daemon, **chuser** command.

The **auditproc** subroutine.

To see the steps you must take to establish an Auditing System, refer to How to Set Up an Auditing System in *General Concepts and Procedures*.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

dir File

Purpose

Describes the format of a directory.

Description

A directory is a file that contains information and structures necessary to define a file hierarchy. A file is interpreted as a directory by the system if it has the **S_IFDIR** file mode. All modifications to the structure of a directory must be performed under control of the operating system.

The AIX directory file format accommodates component names of up to 256 characters. This is accomplished through the use of a variable–length structure to describe individual directory entries. The structure of a directory entry is:

Note: This structure is an AIX file system specific data structure. It is recommended that file system independent application programs use the file system independent direct structure and its associated library support routines.

```
struct direct {
        ino_t     d_ino;
        ushort    d_reclen;
        ushort    d_namelen;
        char      d_name[256];
  };
```

By convention, the first two entries in each directory are . (dot) and .. (dot dot). The . (dot) is an entry for the directory itself. The .. (dot dot) entry is for the parent directory. Within the root (/) directory the meaning of .. (dot dot) is modified; because there is no parent directory, the .. (dot dot) entry has the same meaning as the . (dot) entry.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/sys/dir.h The path to the **dir.h** header file.

Related Information

The **fs** file format, **inode** file format, **dirent.h** file format.

The **stat** command.

The **opendir** directory subroutines, **readdir** directory subroutines, **telldir** directory subroutines, **seekdir** directory subroutines, **closedir** directory subroutines.

The File Systems Overview in *General Concepts and Procedures* explains file system types, management, structure, and maintenance.

The Directories Overview in *General Concepts and Procedures* explains working with directories and path names.

The Files Overview in *General Concepts and Procedures* provides information on working with files.

distfile File

Purpose

Contains directions for the **rdist** command.

Description

The **distfile** file contains commands for the **rdist** command. The **rdist** command maintains identical copies of files on multiple hosts. The entries in the **distfile** file specify the files to be copied, destination hosts to distribute files to, and operations to perform for updating files to be distributed with the **rdist** command.

Each entry in the **distfile** file has one of the following formats:

```
VariableName = NameList
```

```
[Label:] SourceList -> DestinationList CommandList
```

```
[Label:] SourceList :: TimeStampFile CommandList
```

Define variables with the first format. Use the second format to distribute files to other hosts. Use the third format to make lists of those files that have changed since a given date. The *SourceList* variable specifies a list of files and directories on the local host for the **rdist** command to use as the master copy for distribution. The *DestinationList* variable is the list of hosts to receive copies of the files.

Each file in the list specified by the *SourceList* variable, indicated with the second format, is updated if the file is out-of-date on the host being updated. Each file specified with the *SourceList* variable, indicated with the third format, is updated if the file is newer than the timestamp file. The third format is useful for restoring files.

Labels are optional and are used to identify a command for partial updates.

The **rdist** command treats new-line characters, tabs, and blanks as separators. Variables for expansion begin with a $ (dollar sign) followed by a single character or a name enclosed in {} (braces). Comments begin with a # (pound sign) and end with a new-line character.

Source and Destination List Format

The source and destination lists have the following format:

```
Name
```

or

```
(Zero or more names separated by blanks)
```

The shell metacharacters [,], {, }, (,), *, and ? (8 different characters) are recognized and expanded on the local host in the same way as for the **csh** command. Keep them from being expanded with a \ (backslash). The **rdist** command also expands the ~ (tilde) n the same way as for the **csh** command, but does so separately on the local and destination hosts.

When the **rdist** command flag **–w** is used with a file name that begins with a tilde, everything except the home directory is appended to the destination name. File names that do not begin with a \ (backslash) or a ~ (tilde) use the destination user's home directory as the root directory for the rest of the file name.

Command List

The command list consists of zero or more of the following commands:

install *Options* [*OptionalDestName*] ;

The **install** command copies out-of-date files and directories. The **rdist** command copies each source file or directory to each host in the destination list. The available options as specified by the *Options* variable are the **rdist** command flags **–b**, **–h**, **–i**, **–R**, **–v**, **–w**, and **–y**. These options only apply to the files in the *SourceList*. When you use the **–R** flag, non-empty directories are removed if they are symbolic links. The *OptionalDestName* variable renames files.

If no **install** command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name are created if they do no exist on the remote host. The login name used on the destination host is the same as the local host unless the destination name is of the format `login@host`.

notify *NameList* ;

The **notify** command mails the list of updated files to the listed names (the *NameList* variable). If no @ (*at* sign) appears in the name, the destination host is appended to the name (`name@host`).

except *NameList* ;

The **except** *NameList* command causes the **rdist** command to update all the files in the *SourceList* except for the files listed in the *NameList*.

except *PatternList* ;

The **except** *PatternList* command prevents the **rdist** command from updating any files that contain a string that matches a member of the list specified by the *PatternList* variable.

special *NameList* "*String*" ;

The **special** command specifies shell commands (the "*String*" variable) to be executed on the remote host after the file in the *NameList* variable is updated or installed. If the *NameList* variable is omitted, the shell commands are executed for every file updated or installed. The shell variable *File* is set to the current file name before the **rdist** command executes the "*String*" variable. The "*String*" variable can cross multiple lines in the distribution file.

Multiple commands to the shell must be separated by a ; (semicolon). Commands are executed in the user's home directory on the host being updated. The **special** command can be used to rebuild private databases after a program has been updated.

Examples

Examples of the format `VariableName = NameList`

1. To indicate which hosts' files to update, enter a line similar to the following:

```
HOSTS =( matisse root@arpa )
```

In this example the variable name is `HOSTS`, defined to be `matisse` and `root@arpa`. The **rdist** command will update files on the hosts `matisse` and `root@arpa`. This variable could be used as a destination list.

2. To indicate a name to use as a value for a *SourceList* variable, enter a line similar to the following:

```
FILES = ( /bin /lib/usr/bin /usr/games
    /usr/include/{*.h{stand,sys,vax*,pascal,machine}/*.h}
       /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )
```

   In this example, the varable name is `FILES`, defined as the files to be used for either the *SourceList* varable.

3. To indicate which files to exclude from the updating process, enter a line similar to the folowing:

```
EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
    sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont)
```

   In this example, the variable is `EXLIB`, defined as a list of files to exclude from the updating process.

Examples of the format `[label:] SourceList -> DestinationList CommandList`

4. To copy a source list of files to a destination list of hosts, enter a line similar to the following:

```
${FILES} ->${HOSTS}
    install -R
    except /usr/lib/${EXLIB}  ;
    except /usr/games/lib  ;
    special /usr/lib/sendmail "/usr/lib/sendmail.bz"  ;
```

   The **[label:]** portion of the line is optional and is not shown here. The `$` (dollar sign) and the {} (braces) cause the names `FILES`, `HOSTS`, and `EXLIB` to be expanded into the lists designated for them in the previous examples. The rest of the example comprises the command list.

5. To use the **[label:]** portion, enter the line as follows:

```
srcsL:
/usr/src/bin -> arpa
    except_pat (\e\e.o\e$ /SCCS\e$ ) ;
```

   The label is `srcsL:` and and can be used to identify this entry for updating. The `/usr/src/bin` file is the source to be copied and the host `arpa` is the destination of the copy. The third line contains a command from the command list.

6. To use a timestamp file, enter a line similar to the following:

```
${FILES} :: stamp.cory
    notify root@cory
```

   The `$` (dollar sign) and `{}` (braces) cause the name `FILES` to be expanded into the list designated for it in example 2. The time stamp file is `stamp.cory`. The last line is a command from the command list.

Files

$HOME/distfile	Contains commands for the **rdist** command.
/tmp/rdist	Temporary file for update lists.

Related Information

The **csh** command, **rdist** command, **stat** command.

dumpdates File

Purpose

Describes the format of the **dumpdates** file.

Description

The **dumpdates** file holds date information for the **backup** command and **rdump** command. The following is the **dumpdates** file structure:

```
struct  idates {
        char    id_name[MAXNAMLEN+3];
        char    id_incno;
        time_t  id_ddate;
```

The `struct idates` describes an entry in the **/etc/dumpdates** file where the backup history is kept. The fields of the structure are:

id_name The name of the file system (referred to as the **/dev/id_nam** file).

id_incno The level number of the backup tape.

id_ddate The date of the incremental backup in system format.

MAXNAMLEN The maximum is 255.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/etc/dumpdates Specifies the location of the command.

Related Information

The **backup** command, **rdump** command.

The Backup Overview in *General Concepts and Procedures* provides information on different methods of backing up, restoring process, different types of backup media, and guidelines for backup policies.

emaltdefs.p File

Purpose

Contains alternate default keyboard layout, screen colors, and field attribute modes used in 3278/79 Terminal Emulation.

Description

The **/usr/lib/em78/emaltdefs.p** file contains the alternate default settings used by the EM78 Emulation program. (The standard default settings are defined in the **/usr/lib/em78/emdefs.p** file.) The settings in the **emaltdefs.p** file affect the following aspects of the emulation:

- Key Definitions
- Screen Colors
- Field Attribute Colors and Modes.

A copy of the **emaltdefs.p** file can be modified to change the key, color, or attribute settings. The modified file is then installed using the **emkey** command. The **emaltdefs.p** file is an example of an EM78 customization file.

Note: It is recommended that you edit a *copy* of the **emaltdefs.p** file and keep the original intact in case you want to return the emulator settings to the original alternate settings.

Implementation Specifics

This file is part of 3278/79 Emulation in AIX 3278/79 Emulation/6000.

File

/usr/lib/em78/emdefs.p	Contains default keyboard settings for EM78.

Related Information

The **em78** command, **emkey** command.

EM78 Customization File Format.

EM78 Overview for Managers in *Communication Concepts and Procedures*.

EM78 Field Attribute Codes in *Communication Concepts and Procedures*.

emdefs.p File

Purpose

Contains the default keyboard layout, screen colors, and field attribute modes used in 3278/79 Terminal Emulation.

Description

The **/usr/lib/em78/emdefs.p** file contains the default settings used by the EM78 Emulation program. These settings affect the following aspects of the emulation:

- Key Definitions
- Screen Colors
- Field Attribute Colors and Modes.

A copy of the **emdefs.p** file can be modified to change the key, color, or attribute settings. The modified file is then installed using the **emkey** command. The **emdefs.p** file is an example of an EM78 customization file.

Note: It is recommended that you edit a *copy* of the **emdefs.p** file and keep the original intact in case you want to return the emulator settings to the default settings.

Implementation Specifics

This file is part of 3278/79 Emulation in AIX 3278/79 Emulation/6000.

File

/usr/lib/em78/emaltdefs.p	Contains alternate configuration settings for EM78 which can be used in place of the default settings in the **emdefs.p** file.

Related Information

The **em78** command, **emkey** command.

EM78 Customization file format.

EM78 Overview for Managers in *Communication Concepts and Procedures.*

environ File

Purpose

Defines the environment attributes for users.

Description

The **/etc/security/environ** file is an ASCII file that contains stanzas with the environment attributes for users. Each stanza is identified by a user name and contains attributes in the *Attribute*=V*alue* form, with a comma separating the attributes. Each attribute is ended by a new line character, and each stanza is ended by an additional new line character. For an example of a stanza, see the Examples section.

If environment attributes are not defined, the system uses default values. Each user stanza can have the following attributes:

usrenv Defines variables to be placed in the user's environment when the initial **login** command is given or when the **su** command resets the environment. The value is a list of comma–separated attributes. The default value is an empty string.

sysenv Defines variables to be placed in the user's protected state environment when the initial **login** command is given or when the **su** command resets the environment. These variables are protected from access by unprivileged programs so other programs can depend on their values. The default value is an empty string.

For a description of environment variables, refer to the **environment** File.

Access to all the user database files should be through the system commands and subroutines defined for this purpose. Access through other commands or subroutines may not be supported in future releases.

The **mkuser** command creates a user stanza in this file. The initialization of the attributes depends upon their values in the **/etc/security/mkuser.default** file. The **chuser** command can reset these attributes, and the **lsuser** command can display them. The **rmuser** command removes the entire record for a user. For programming information, see the system calls in the Related Information section.

Security

Access Control: This command should grant read (r) access to the root user, members of the security group, and others consistent with the security policy for the system. Only the root user should have write (w) access.

Auditing Events:

Event	Information
S_ENVIRON_WRITE	filename

Example

1. A typical stanza looks like the following example for user `dhs:`

```
dhs:
   usrenv = "MAIL =/u/spool/mail/dhs"
   sysenv = "NAME = dhs@delos"
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/environ	Specifies the path to the file.
/etc/environment	Specifies the basic environment for all processes.
/etc/group	Contains the basic attributes of groups.
/etc/security/group	Contains the extended attributes of groups.
/etc/passwd	Contains the basic attributes of users.
etc/security/passwd	Contains password information.
etc/security/user	Contains the extended attributes of users.
etc/security/limits	Contains the process resource limits of users.
/etc/security/mkuser.default	Contains the default values for user accounts.

Related Information

The **chuser** command, **login** command, **lsuser** command, **mkuser** command, **rmuser** command, **setsenv** command, **su** command.

The **getpenv** subroutine, **getuserattr** subroutine, **putuserattr** subroutine, **setpenv** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

environment File

Purpose

Sets up the user environment.

Description

When a new process begins, the **exec** subroutine 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.

The **exec** subroutine allows the entire environment to be set at one time. Ensure newly created environment variables do not conflict with standard variables 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 your login profile, **.profile**, is read. The following variables make up the basic environment:

HOME The full path name of the users login or HOME directory. The **login** program sets this to the name specified in the **/etc/passwd** file.

LANG The locale name currently in effect. The **LANG** variable is set in the **/etc/profile** file at installation time. The default value is LANG=EN_US.pc850.

NLSPATH The full path name for message catalogs. The default is **/usr/lpp/msg/%L/%N: /usr/lpp/msg/prime/%N** where **%L** is the value of the **LANG** variable and **%N** is the catalog file name.

LOCPATH The full path name of the location of National Language Support tables. The default is **/usr/lib/nls** and is set in the **/etc/profile** file. If the **LOCPATH** variable is null, it assumes that the current directory contains the locale files.

PATH The sequence of directories that commands such as the **sh**, **time**, **nice** and **nohup** commands search when looking for a command whose path name is incomplete. The directory names are separated by colons.

TZ The time zone information. The **TZ** environment variable is set by the **/etc/profile** file, the system login profile. The **TZ** environment variable has the following format:

```
std offset dst offset , rule
```

std and *dst* Only *std* is required. If *dst* is not specified, summer time does not apply. Uppercase and lowercase letters are allowed in the designation. Any characters except a : (colon), digits, , (comma), – (minus), + (plus) and ASCII NULL are allowed. Three or more bytes are the designation for *std* (standard time zone) and for *dst* (summer time zone).

offset Indicates the value added to local time to equal Coordinated Universal Time. The *offset* variable has the following format:

```
hh[:mm[:ss]]
```

The *offset* variable following the *std* variable is required. If the *offset* variable does not follow the *dst* variable, summer time is assumed to be one hour ahead of standard time. One or more digits can be used; interpret as a decimal number. The hour must be specified between 0 through 24. The *mm* (minutes) variable and *ss* (seconds) variable are optional. If these variables are present, they must be specified between 0 and 59. If the variable is preceded by a – (minus), the time zone is east of the Prime Meridian. If the time zone variable is not preceded by a – (minus), the time zone is assumed to be west of the Prime Meridian. The time zone can be preceded by a + (plus) to indicate it is west of the Prime Meridian.

rule The rule variable has the following format:

```
date/time,date/time
```

The *date* variable describes when the change from standard to summer time occurs. The second *date* variable describes when the change back to standard occurs. Each time variable describes, in current local time, when the change is made.

The *date* variable has the following format:

```
Jn or Mm.n.d
```

J*n* where J is the Julian day of the year. The *n* variable has the value of 1 through 365. Leap days are not counted.

M*m.n.d* where the *m* variable is the month, the *n* variable is the week, and the *d* variable is the day of the month. Week one is the first week when the *d* day occurs. Day zero is Sunday.

The *time* variable has the same format as the *offset* variable except no leading sign – (minus) or + (plus) is allowed. The default of the *time* variable is 02:00:00.

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/profile	Specifies variables to be added to the environment by the shell.
/etc/environment	Specifies the basic environment for all processes.
$HOME/.profile	Specifies the environment for a specific users needs.
/etc/passwd	Specifies user IDs.

Related Information

The **ctab** command, **env** command, **export** command, **login** command, **sh** command, **at** command, **getty** command.

The **exec** subroutine, **getenv** subroutine.

eqnchar File

Purpose

Contains special character definitions for the **eqn** command and **neqn** command.

Description

The **/usr/pub/eqnchar** file contains the following **troff** command and **nroff** command character definitions that are not ordinarily available on a phototypesetter or printer. These definitions are primarily intended for use with the **eqn** command and **neqn** command. The **eqnchar** file contains definitions for these characters.

The **/usr/pub/cateqnchar** file is device-independent and should produce output that looks reasonable on any device supported by the **troff** command. You may link the **/usr/pub/eqnchar** file to the **/usr/pub/cateqnchar** file.

The **eqnchar** file format can be used with the **eqn** command or **neqn** command and piped to the **troff** command or **nroff** command. For example:

eqn /usr/pub/eqnchar [*Flag(s)*] [—] [*File(s)*] | **troff** [*Flag(s)*]

eqn /usr/pub/cateqnchar [*Flag(s)*] [—] [*File(s)*] | **troff** [*Flag(s)*]

neqn /usr/pub/eqnchar [*Flag(s)*] [—] [*File(s)*] | **nroff** [*Flag(s)*]

Implementation Specifics

This file is part of Formatting Tools in the Text Formatting System of AIX for RISC System/6000.

Files

/usr/pub/eqnchar	**nroff** and **troff** character definitions file.
/usr/pub/cateqnchar	Character definitions for **troff**-supported device.

Related Information

The **eqn** command, **neqn** command, **nroff** command, **troff** command, **mm** command, **mmt** command, and **mvt** command.

events File

Purpose

Contains information about system audit events.

Description

The **/etc/security/audit/events** file is an ASCII stanza file that contains information about audit events. The file contains just one stanza, auditpr, which lists all the audit events in the system. The stanza also contains formatting information that the **auditpr** command needs to write an audit tail for each event.

Each attribute in the stanza is the name of an audit event, with the following format:

AuditEvent = *FormatCommand*

The format command can have the following parameters:

(empty)	The event has no tail.
printf *Format*	The tail is formatted according to the string supplied for the *Format* parameter. The %x symbols within the string indicate places for the audit trail to supply data. See Example for an illustration.
Program **–i** *n Arg*	The tail is formatted by the program specified by the *Program* parameter. The **–i** *n* parameter is passed to the program as its first parameter, indicating that the output is to be indented by *n* spaces. Other formatting information can be specified with the *Arg* parameter. The audit event name is passed as the last parameter. The tail is written to the standard input of the program.

Security

Access Control: This file should grant read (r) access to the root user and members of the audit group, and grant write (w) access only to the root user.

Example

To format the tail of an audit record for new audit events, such as `FILE_Open`, `PROC_Create`, and `login`, add format specifications like the following to the auditpr stanza in the **/etc/security/audit/events** file:

```
auditpr:
  FILE_Open = printf "flags: %d mode: %d \
   fd: %d name: %s"
  PROC_Create = printf "child: %d"
  login = printf "name: %s"
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/audit/events	Specifies the path to the file.
/etc/security/audit/config	Contains audit system configuration information.
/etc/security/audit/objects	Contains information about audited objects.
/etc/security/audit/bincmds	Contains auditbin backend commands.
/etc/security/audit/streamcmds	Contains auditstream commands.

Related Information

The **auditpr** command.

To see the steps you must take to establish an Auditing System, refer to How to Set Up an Auditing System in *General Concepts and Procedures*.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

filesystems File

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 logical volume. 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 stanza names that are file system names and contents that 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 file system attributes specify all the parameters of the file system. 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 the **true** or **false** value.

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. The **true** value enables checking while the **false** value disables checking. If a number, rather than the **true** value is specified, the file system is checked in the specified pass of checking. Multiple pass checking, described in the **fsck** command, permits file systems on different drives to be checked in parallel.

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, it identifies the file or directory to be mounted.

mount Used by the **mount** command to determine whether this file system should be mounted by default. The possible values of the **mount** attribute are:

automatic Automatically mounts a file system when the system is started. For example, in the sample file, the root file system line is the **mount=automatic** attribute. This means that the root file system mounts automatically when the system is started. The **true** value is not used so that **mount all** does not try to mount it, and **umount all** doesn't try to unmount

it. Also, it is not the **false** value because certain utilities, such as the **ncheck** command, normally avoid file systems with a value of the **mount=false** attribute.

false This file system is not mounted by default.

readonly This file system is mounted as read-only.

true This file system is mounted by the **mount all** command. It is unmounted by the **umount all** command. The **mount all** command is issued during system initialization to automatically mount all such file systems.

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 the **nodename** attribute should be a valid node nickname. This value can be overridden with the **mount –n** command.

size Used by the **mkfs** command for reference and for building the file system. The value is the number of 512–byte blocks in the file system.

type Used to group related mounts. When the **mount –t** *String* command is issued, all of the currently unmounted file systems with a **type** attribute equal to the *String* parameter are mounted.

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 6 characters.

log The device to which log data is written as this file system is modified. This is only valid for journaled file systems.

Example

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

Warning: Modifying this file can cause several effects to file systems.

```
*
* File system information
*
default:
        vol         = "AIX"
        mount       = false
        check       = false

/:
        dev         = /dev/hd4
        vol         = "root"
        mount       = automatic
        check       = true
        log         = /dev/hd8

/u:
        dev         = /dev/hd1
        vol         = "u"
        mount       = true
        check       = true
        log         = /dev/hd8
```

```
/u/joe/1:
        dev         = /u/joe/1
        nodename    = vance
        vfs         = nfs
/usr:
        dev         = /dev/hd2
        vol         = "usr"
        mount       = true
        check       = true
        log         = /dev/hd8
/tmp:
        dev         = /dev/hd3
        vol         = "tmp"
        mount       = true
        check       = true
        log         = /dev/hd8
```

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/etc/filesystems	Lists the known file systems and defines their characteristics.
/etc/vfs	Contains descriptions of virtual file system types.

Related Information

The **fs** file format.

The **backup** command, **df** command, **fsck** command, **mkfs** command, **mount** command, **restore** command, **umount** command, **dodisk** command.

The File Systems Overview in *General Concepts and Procedures* explains file system types, management, structure, and maintenance.

The Mounting Overview in *General Concepts and Procedures* explains mounting files and directories, mount points, and automatic mounts.

The Directories Overview in *General Concepts and Procedures* explains working with directories and path names.

The Files Overview in *General Concepts and Procedures* provides information on working with files.

The Logical Volume Storage Overview in *General Concepts and Procedures* explains the Logical Volume Manager, physical volumes, logical volumes, volume groups, organization, ensuring data integrity, and understanding the allocation characteristics.

fs File

Purpose

Contains the format of a file system logical volume.

Description

A file system storage volume has a common format for vital information. The file system is comprised of a number of 4096-byte logical blocks, however, the superblock contains a size field that is maintained in 512 byte blocks.

A unique feature of the journaled file system is the implementation of file system meta-data, as unnamed files that reside in that file system. For example, the disk i-nodes for any file system are contained in the blocks allocated to the file described by the **INODES_I** i-node number. The i-node number for the boot file is i-node 0. Each of the following reserved i-node numbers corresponds to a file system meta-data file and serves the purpose given:

SUPER_I Superblock file.

INODES_I Disk i-nodes.

INDIR_I Indirect file blocks, double and single.

INOMAP_I I-node allocation bit map.

ROOTDIR_I Root directory i-node.

DISKMAP_I Block allocation bit map.

INODEX_I I-node extensions.

INODEXMAP_I Allocation map for i-node extensions.

A volume is divided into a number of 4096-byte logical blocks. The 512-byte unit term refers to a cluster of one or more such units.

The logical block 0 is unused and available to contain a bootstrap program or other information. The logical block 1 is the file system superblock. The structure of a native-format file system superblock follows:

```
/* The following disk-blocks are formatted or reserved:
 *
 *      ipl block 0 - not changed by filesystem.
 *
 *      superblocks at block 1 (primary superblock)
 *      and block 31 (secondary superblock)
 *      the secondary super-block location is
 *      likely to be on a different disk-surface
 *      than the primary super-block. both blocks
 *      are allocated as blocks in ".superblock".
```

```
*
 *      blocks for .inodes according to BSD layout.
 *      the first allocation group uses disk-blocks
 *      number 32 and the next consecutive blocks
 *      sufficient for one inode per disk-block in
 *      the allocation group. the inode blocks
 *      for all other allocation groups start in the
 *      first block of the allocation group and continue
 *      in consecutive blocks sufficient for one inode
 *      per disk block of the allocation group.
*
 *      other disk-blocks are allocated for .indirect,
 *      .diskmap, .inodemap, and their indirect blocks
 *      starting in the first allocation-group.
/*
 *The special fs inodes formatted and their usage is as follows:
 *
 *      inode 0 - never allocated - reserved by setting n_link = 1
 *      inode 1 - inode for .superblock
 *      inode 2 - inode for root directory
 *      inode 3 - inode for .inodes
 *      inode 4 - inode for .indirect
 *      inode 5 - inode for .inodemap - allocation map for .inodes
 *      inode 6 - inode for .diskmap - disk allocation map
 *      inode 7 - inode for .inodex - inode extensions
 *      inode 8 - inode for .inodexmap - allocation map for .inodex
 *      inode 9 - 16 - reserved for future extensions
*
 *except for the root directory, the special inodes are not in any
 *directory.
 *
*/#define IPL_B          0
 #define SUPER_B        1
 #define SUPER_B1       31
 #define INODES_B       32
 #define NON_B          0
 #define SUPER_I        1
 #define ROOTDIR_I      2
 #define INODES_I       3
 #define INDIR_I        4
 #define INOMAP_I       5
 #define DISKMAP_I      6
 #define INODEX_I       7
 #define INDOESMAP_I    8
/*
 * super block format. primary superblock is located in block
 * (page) one.
 * the secondary super-block is not used except for disaster
 * recovery.
```

```
*/
struct superblock
{
   char s_magic[4];     /*magic number: fsv3magic = 0x43218765 */
   char s_flag[4];      /*flag word (see below)
   int  s_agsize;       /*size of allocation group in pages */
   int  s_logserial; /*serial number of log when fs mounted */
   daddr_t s_fsize;     /*size (in 512 bytes) of entire file system
*/

   short s_bsize;       /*block size (in bytes) for this system */
   short s_spare;       /* unused.                                 */
   char  s_fname[6]; /* name of this file system      */
   char  s_fpack[6]; /* name of this volume           */
   dev_t s_logdev;      /* device address of log         */

/* current file system state information, values change over time */
char     s_fmod;         /* flag: set when file system is mounted */
char     s_ronly;        /*flag: file system is read only */
time_t   s_time;         /* time of last superblock update      */
};
```

This file actually lives in **/usr/include/jfs/filsys.h**. But, if **/usr/include/sys/filsys.h** is included, you will get the **jfs/filsys.h** file included by default.

The superblock's magic number is encoded as a 4-byte character string to make it possible to validate the superblock without knowing the byte order of the remaining fields. To check for a valid superblock, enter something like:

```
if (strncmp(sp->s_magic,FSv3magic,4) == 0)
```

The last byte of the **s_flag** field gives the blocksize-dependent information. The first byte of the **s_flag** field gives the CPU code for the host system with the byte order encoded in the low-order bits.

```
#define fsv3magic "\102\041\207\145"
#define s_cpu   s_flag[0]       /* Target cpu type code */
#define s_type s_flag[3]        /* File system type code */
```

The fields of the AIX superblock structure follow:

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

The **s_fmod** field is a flag to indicate the cleanliness of the file system. Whenever a file system is mounted, this flag is checked and a warning message is printed if the **s_fmod** field is nonzero. A file system whose **s_fmod** field is 0 is very likely to be clean, and a file system whose **s_fmod** field is 2 is likely to have problems. The **s_fmod** 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 clean and mounted.
- **2** = File system was mounted when dirty.

If you merely mount and unmount the file system, the flag toggles back and forth between states 0 and 1. If you mount the file system while the flag is in state 1, it goes to state 2 and stays there until you run the **fsck** command. The only way to clean up a corrupted file system (change the flag from state 2 back to state 0) is to run the **fsck** command.

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

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

The i-node numbers begin at 1, and the storage for the i-node number 1 begins in the first byte of block 2.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/jfs/filsys.h	The path to the **jfs/filsys.h** header file.
/usr/include/sys/filsys.h	The path to the **filsys.h** header file, which contains an **#include** statement for the **jfs/filsys.h** header file.

Related Information

The **param.h** file format, **i-node** file format.

The **fsck** command, **fsdb** command, **mkfs** command.

The File Systems Overview in *General Concepts and Procedures* explains file system types, management, structure,

The Mounting Overview in *General Concepts and Procedures* explains mounting files and directories, mount points, and automatic mounts.

The Logical Volume Storage Overview in *General Concepts and Procedures* explains the Logical Volume Manager, physical volumes, logical volumes, volume groups, organization, ensuring data integrity, and understanding the allocation characteristics.

gps File

Purpose

Used as the format for storing graphic file data as a graphic primitive string (GPS).

Description

The **gps** file format is used as the format for storing graphic file data as a graphic primitive string (GPS). The **plot** and **vtoc** commands produce GPS output files. Several commands edit and display GPS files on various devices.

Types of Graphic Data or Primitives

A GPS is composed of as many as five types of graphic data or primitives. 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 4 bits of data.

The following are the types of primitive strings and their parameters:

Comment Primitives

A Comment primitive does not cause anything to be displayed. The Comment primitive is an integer string. All GPS files begin with a comment of zero length.

Comment *ControlWord* [*String*]:
The *ControlWord* parameter is the control word. The first 4 bits identify the Comment primitive and have the value 0xF. The following bits give the command word count for the primitive.

The [*String*] parameter 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 Primitives

The Lines primitive contains a variable number of points from which zero or more connected line segments are produced. The first point relocates the graphics cursor to that point, without drawing. Successive points produce line segments starting at the initial point.

Lines *ControlWord Points StyleWord*:
The *ControlWord* parameter is the control word. The first 4 bits identify the Lines primitive and have the value 0x0. The remaining bits give the command word count for the primitive.

The *Points* parameter consists of 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).

The *StyleWord* parameter is the style command word. The first 8 bits hold an integer value for color information. The next 4 bits contain an integer value for weight to indicate line thickness:

0	Narrow
1	Bold
2	Medium.

The last 4 bits of the *StyleWord* parameter specify an integer value giving linestyle information:

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

Arc Primitives

The Arc primitive contains 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 results. 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.

Arc *ControlWord Points StyleWord*:
The *ControlWord* parameter is the control word. The first 4 bits identify the Text primitive and have the value 0x2. The remaining 12 bits contain the command word count for the primitive.

The *Points* parameter is a pair of integer coordinates that are a value within a Cartesian plane or universe of 65,536 points on each axis (–32,767 to +32,768).

The *StyleWord* parameter is the style command word. The first 8 bits hold an integer value for color information. The next 4 bits contain an integer value for weight to indicate line thickness:

0	Narrow
1	Bold
2	Medium.

The last 4 bits of the *StyleWord* parameter specify an integer value giving linestyle information:

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

Text Primitives

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

Text *ControlWord Point FontCommandWord SizeOrientationWord* [*string*]:

The *ControlWord* parameter is the control word. The first 4 bits identify the Text primitive and have the value 0x2. The remaining 12 bits contain the command word count for the primitive.

The *Points* parameter is a pair of integer coordinates that are a value within a Cartesian plane or universe of 65,536 points on each axis (–32,767 to +32,768).

The *FontCommandWord* parameter is a font command word. The first 8 bits contain an integer value for color information. The next 8 bits contain an integer value for font information, with 4 bits giving the weight (density) value for the font, and 4 bits giving the style (typeface) value for the font.

The *SizeOrientationWord* parameter indicates the size or orientation of a command word. Eight bits specify text size as an integer value that indicates the size of characters drawn. Text size represents character height in absolute universe units.

The actual character height is five times the text size value. The next 8 bits are a signed integer value for text angle, and express the angle and direction of rotation of the character string around the beginning point. The text angle is expressed in degrees from the positive X axis. The text angle value is 256/360 of its absolute value.

The Hardware Primitive

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.

Hardware *ControlWord Point* [*String*] :

The *ControlWord* parameter is the control word. The first 4 bits identify the hardware primitive and have the value 0x4. The next 12 bits indicate the command word count for the primitive.

The *Point* parameter 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). The specified point becomes the starting point for the *String* parameter, which is a string of hardware characters or control commands to a hardware device.

File

/usr/include/sys/stat.h

The path to the **stat.h** header file, which defines the data structure used by the status subroutines.

Related Information

The **stat** command and **ttoc** commands.

/etc/group File

Purpose

Contains basic group attributes.

Description

The **/etc/group** file is an ASCII file that contains the basic group attributes. Each record is identified by a group name, and contains the following colon-separated attributes:

Name	The unique name by which a group is known on the system. The name specified by the *Name* parameter must be an alphanumeric string of 8 characters or less that begins with an alphabetic character, and cannot be the **ALL** or **default** `keywords`.
Password	Not used. Group administrators are provided instead of group passwords.
ID	The ID of the group. The value is a unique decimal integer string.
Users	The members of the group, specified as a list of comma-separated user names. Each user must be defined in the local database configuration files or by a credentials server.

Note: Since the colon character is a field separator, it may not be used as part of an attribute.

For an example of a record, see the Example section below.

Additional attributes are defined in the **/etc/security/group** file.

To change the *Name* parameter, you need to add a new entry (with the **mkgroup** command) and remove the old one (with the **rmgroup** command). To display all the attributes in the file, use the **lsgroup** command.

Users can be changed with the **chgroup** command, the **chgrpmem** command, and the **chuser** command. The **mkuser** command adds a user whose primary group is defined in the **/etc/security/mkuser.default** file and the **rmuser** command removes a user. Although the group ID can be changed with the **chgroup** command, this is not recommended.

Security

Access Control: This file should grant read (r) access to all users and grant write (w) access only to the root user and members of the security group.

Example

1. A typical record looks like the following example for the `staff` group:

   ```
   staff::1:shadow,cjf
   ```

   In this example, the *GroupID* is `1` and `shadow` and `cjf` are *Users*. The *Password* field is not used.

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/group	The path to the file.
/etc/security/group	Contains the extended attributes of groups.
/etc/passwd	Contains the basic attributes of users.
/etc/security/passwd	Contains password information.
/etc/security/user	Contains the extended attributes of users.
/etc/security/environ	Contains the environment attributes of users.
/etc/security/limits	Contains the process resource limits of users.
/etc/security/audit/config	Contains audit system configuration information.

Related Information

The **chfn** command, **chgrpmem** command, **chsh** command, **chgroup** command, **chuser** command, **lsgroup** command, **lsuser** command, **mkgroup** command, **mkuser** command, **passwd** command, **pwdadm** command, **rmgroup** command, **rmuser** command, **setgroups** command, **setsenv** command.

The **getgroupattr** subroutine, **putgroupattr** subroutine, **IDtogroup** subroutine, **nextgroup** subroutine, **setuserdb** subroutine, **enduserdb** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

/etc/security/group File

Purpose

Contains extended group attributes.

Description

The **/etc/security/group** file is an ASCII file that contains stanzas with extended group attributes. Each stanza is identified by a group name from the **/etc/group** file followed by a colon, and contains attributes in the form *Attribute=Value*. Each attribute is ended with a new line character and each stanza is ended with an additional new line character. For a typical stanza, refer to the Example section below.

Each stanza can have either or both of the following attributes:

adms Defines the users who can perform administrative tasks for the group, such as setting the members and administrators of the group. This attribute is ignored if **admin = true**, since only the root user can alter a group defined as administrative. The value is a list of comma-separated user login-names. The default value is an empty string.

admin Defines the administrative status of the group. Possible values are:

- **true** Defines the group as administrative. Only the root user can change the attributes of groups defined as administrative.
- **false** Defines a standard group. The attributes of these groups can be changed by the root user or a member of the security group. This is the default value.

Access to this file should be through the system commands and subroutines defined for this purpose.

The **mkgroup** command adds new groups to the **/etc/group** file and the **/etc/security/group** file. This command can identify the group as administrative and set the invoker of the command as the administrator, initializing the **adms** attribute.

Use the **chgroup** command to change all the attributes. If you are defined as an administrator of a specified standard group, you can change the **adms** attribute for that group with the **chgrpmem** command.

The **lsgroup** command displays both the **adms** and the **admin** attributes, and the **rmgroup** command removes the entry from both the **/etc/group** file and the **/etc/security/group** file.

To write shell scripts that affect attributes in the **/etc/security/group** file, use the subroutines listed in Related Information.

Security

Access Control: This file should grant read (r) access to the root user and members of the security group, and to others as permitted by the security policy for the system. Only the root user should have write (w) access.

Auditing Events:

Event	Information
S_GROUP_WRITE	filename

Example

A typical stanza looks like the following example for the `finance` group:

```
finance:
          adms = cjf
          admin = false
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/group	The path to the file.
/etc/group	Contains the basic attributes of groups.
/etc/passwd	Contains the basic attributes of users.
/etc/security/passwd	Contains password information.
/etc/security/user	Contains the extended attributes of users.
/etc/security/environ	Contains the environment attributes of users.
/etc/security/limits	Contains the process resource limits of users.
/etc/security/audit/config	Contains audit system configuration information.

Related Information

The **chgroup** command, **chgrpmem** command, **lsgroup** command, **mkgroup** command, **rmgroup** command, **setgroups** command.

The **enduserdb** subroutine, **getgroupattr** subroutine, **IDtogroup** subroutine, **nextgroup** subroutine, **putgroupattr** subroutine, **setuserdb** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

inittab File

Purpose

Controls the initialization process.

Description

The **inittab** file supplies information for the **init** command to dispatch general processes. The **init** command reads the **inittab** file each time the **init** command is invoked. The process that constitutes the majority of the **init** commands process dispatching activities is the line process **/etc/getty** that initiates individual terminal lines. Other processes typically dispatched by the **init** command are daemons and the shell.

The **/etc/inittab** file is composed of lines, each with its own fields, separated by colons. Each line of the **/etc/initab** file looks like the following:

Identifier:*Runlevel*:*Action*:*Command*

There is no limit to the number of lines in the **inittab** file. The fields are:

Identifier This is a fourteen character field used to uniquely identify an object. This field is also used in conjunction with other programs to identify certain objects that are modified by them. Terminal devices might be a six character field `rsnnnn` or `ptsnnn`: where `n` is the number of the device. The device **/dev/tty01** would be denoted by **tty01**, the device **/dev/pts000** by **pts000**. The **/dev/console** device is denoted by **console**.

Runlevel *Runlevel* is a twenty character field which defines the runlevels in which this object is to be processed. *Runlevel* corresponds to a configuration of processes in a system. Each process spawned by the **init** command is assigned one or more runlevels in which it is allowed to exist. The runlevels are represented by characters ranging from **0** to **9**, **S**, **s**, **M or m**.

As an example, if the system is in runlevel **1**, only those objects having a **1** in the *runlevel* field are processed. When the **init** command is requested to change runlevels, all processes which do not have an object in the *runlevel* field for the target runlevels are sent a **SIGTERM** signal and allowed a twenty second grace period before being forcibly stopped by a **SIGKILL** signal. The *runlevel* field can define multiple runlevels for a process by selecting more than one runlevel in any combination of *runlevel* characters. If no runlevel is specified, then the process is assumed to be valid at all runlevels.

Three other values, **a**, **b**, and **c** can appear in this field, even though they are not true runlevels. Entries which have these characters in the *runlevel* field are processed only when the **telinit** command requests them to be run (regardless of the current runlevel of the system). They differ from runlevels in that the **init** command can never enter runlevels **a, b,** or **c**. A request to enter of any of these runlevels does not change the current runlevel. A process started by an **a, b,** or **c** command is not stopped when the **init** command changes levels. They are only stopped if their *Action* field is marked **off**, their objects deleted entirely, or the **init** command goes into the maintenance state. The maintenance state is selected by the *runlevel* characters **S**, **s**, **M** or **m**.

The default **inittab** file is set up so that the default *runlevel* is **2**, which is used as the multiuser level. The console device driver is also set to run at all runlevels so the system can be operated with only the console active.

Action A twenty character field that informs the **init** command how to treat the process specified in the command field. The actions recognized by the **init** command are:

respawn If the process does not exist, start the process. If the process currently exists, then do nothing and continue scanning the **inittab** file.

wait Upon the **init** command entering the runlevel that matches the line's runlevel, start the process and wait for the process to stop. All subsequent reads of the **inittab** file while the **init** command is in the same runlevel will ignore this object.

once Upon the **init** command entering a runlevel that matches this line's runlevel, start the process, do not wait for its termination and when it stops, do not restart the process. If entering a new runlevel, where the process is still running from a previous runlevel change, the program will not be restarted.

boot Process only at the **init** command Initial Program Load (IPL) time read of the **inittab** file. The **init** command starts the process, and does not wait for it to stop. When the process stops, it is not restarted. The runlevel should be the default or it must match the **init** command *runlevel* at boot time.

bootwait Process only at the **init** command IPL time read of the **inittab** file. The **init** command starts the process and waits for it to stop. When it stops, the process is not restarted.

powerfail Execute the process associated with this line only when the **init** command receives a **SIGPWR** signal .

powerwait Execute the process associated with this line only when the **init** command receives a **SIGPWR** signal and wait until it stops before continuing any processing of the **inittab** file.

off If the process associated with this line is currently running, send the **SIGTERM** warning signal and wait 20 seconds before sending the **SIGKILL** signal. If the process is nonexistent, ignore the line.

ondemand This action is really a synonym for **respawn**. It is functionally identical to **respawn** but it is given a different keyword in order to separate its association with runlevels. This is used only with the **a**, **b**, or **c** values described in the *runlevel* field.

initdefault A line with this action is only processed when the **init** command is originally invoked. The **init** command uses this line to determine which runlevel to enter originally. The **init**

command does this by taking the highest runlevel specified in the *runlevel* field and using that as its initial state. If the *runlevel* field is empty, this is interpreted as the **0123456789** value, so the **init** command enters *runlevel* **9**. If the **init** command does not find an **initdefault** line in the **inittab** file, it requests an initial *runlevel* from the operator at IPL time.

sysinit Entries of this type are executed before the **init** command tries to access the console. It is expected that this line is only used to initialize devices on which the **init** command might try to ask the *runlevel* question. These lines are executed before the **init** command continues.

Command This is a 1024 character field which holds the **sh** command to be executed. The entry in the command field is prefixed with exec. Any valid **sh** syntax can appear in the command field. Comments can be inserted with the # comment syntax. The line continuation character \ can be placed at the end of a line.

Examples

1. To work at all runlevels, enter:

   ```
   ident:0123456789:Action:Command
   ```

2. To work only at runlevel 2, enter:

   ```
   ident:2:Action:Command
   ```

3. To disable runlevels 0, 3, 6–9, enter:

   ```
   ident:1245:Action:Command
   ```

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/etc/getty Indicates terminal lines.

Related Information

The **init** command, **telinit** command.

The **kill** subroutine.

inode File

Purpose

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

Description

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

```
struct dinode
 {
        /* generation number */
        ulong di_gen;
        /* the mode_t returned by stat () */
        /* format, attributes and permission bits */
mode_t di_mode;

/* number of links to file (if 0, inode is available) */
ushort di_nlink;

/* accounting ID */
ushort di_acct;

/* user id of owner */
uid_t di_uid;

/* group id of owner */
gid_t di_gid;

/* size of file */
off_t di_gid;

/* number of blocks actually used by file */
ulong di_nblocks;

/* the need for nano-second time stamps is questionable.
 * there is room, but for now the space is just reserved.
*/
/* time last modified */
time_t di_mtime;
ulong di_rsvd1;
/* time last accessed */
time_t di_atime;
ulong di_rsrvd2;
/* time last changed inode */
time_t di_ctime;
ulong di_rsrvd3;

/* extended access control information*/
long di_acl;   /* acl pointer */
#       define ACL_INCORE   (1 << 31)
ulong di_sec;       /*reserved */

/* spares */
ulong di_rsrvd[5];
```

```
/***** file type-dependent information ****/
/*
 * size of private data in disk inode is D_PRIVATE.
 * location and size of fields depend on object type.
 */
#       define D_PRIVATE    48

union di_infor
{
        /* all types must fit within d_private */
        char d_private[D_PRIVATE];
        /* jfs regular file or directory. */
        struct regdir
        {
                /* privilege vector - only for non-directory */
                struct
                {
                        ulong _di_offset;
                        ulong _di_flags;
                        define PCL_ENABLED (1<<31)
                        define PCL_EXTENDED (1<<30)
                        define PCL_FLAGS \
                                (PCL_ENABLED|PCL_EXTENDED)
                }_di_privingo;
                priv_t _di_priv;
                /* ACL templates - only for directory */
                struct
                {
                        ulong _di_aclf;
                        ulong _di_acld;
                        { _di_aclingo;
                } _di_sec;
        } _di_file;

/* offsets of regular file or directory private data. */
#       define  di_rdaddr       _di_info._di_file._di_rdaddr
        define  di_vindirect    _di_info._di_file._di_vinderect
        define  di_rinderect    _di_info._di_file._di_rinderect
        define  di_privinfo   _di_info._di_file._di_sec._di_privinfo
        define  di_privoffset   _di_privinfo._di_privoffset
        define  di_privflags    _di_privinfo._di_privflags
        define  di_priv _di_info._di_file._di_sec._di_priv
        define di_aclf
           _di_info._di_file._di_sec._di_aclinfo._di_aclf
        define  di_acld
          di_info._di_file._di_sec._di_aclinfo._di_acld
                /* special file (device) */
                struct
                }
                   dev_t _di_rdev:      /*device major and minor*/
                   ulong _di_bnlastr;   /*read-ahead..last block*/
                   ino_t _di_pino; /*Parent inode for mpx channels*/
                } _di_dev;

/* offsets of special file private data */
#       define di_rdev  _di_infor._di_dev._di_rdev
#       define di_bnlastr       _di_info._di_dev._di_bnlastr
#       define di_dgp   _di_info._di_dev._di_dgp
#       define di_pino  _di_info._di_dev._di_pino
```

```
                /*
                 * symbolic link. link is stored in inode if its
                 * length is less than D_PRIVATE. Otherwise like
                 * a regular file.
                 */
                union
                {
                        char    _s_private[D_PRIVATE];
                        struct  regdir _s_symfile;
                }_di_sym;
/* offset of symbolic link private data */
#       define di_symlink      _di_info._di_sym._s_private

                /* fifo . this info need not be on disk. */
                struct fifonode
                {
                struct fifo_bufhdr *fn_buf;    /*ptr to first buf */
                struct fifo_bufhdr *fn_bufend; /*ptr to last buf */
                ulong           fn_size       /*bytes in fifo */
                off_t           fn_wptr;      /* write offset */
                off_t           fn_rptr;      /*read offset */
                ushort          fn_poll;      /*requested events */
                short           fn_rcnt;      /* # waiting readers*/
                short           fn_wcnt;      /* waiting writers */
                short           fn_flag;      /* (see below) */
                short           fn_nbuf;      /* # bufs allocated */
                }_di_fifo;

/* offsets of FIFO data */
#       define  di_ifn_buf      _di_info._di_fifo.fn_buf
#       define  di_ifn_bufend   _di_info._di_fifo.fn_bufend
#       define  di_ifn_poll     _di_info._di_fifo.fn_poll
#       define  di_ifn_size     _di_info._di_fifo.fn_size
#       define  di_ifn_wcnt     _di_info._di_fifo.fn_wcnt
#       define  di_ifn_rcnt     _di_info._di_fifo.fn_rcnt
#       define  di_ifn_wptr     _di_info._di_fifo.fn_wptr
#       define  di_ifn_rptr     _di_info._di_fifo.fn_rptr
#       define  di_ifn_flag     _di_info._di_fifo.fn_flag
#       define  di_ifn_nbuf     _di_info._di_fifo.fn_nbuf
        /*
         * data for mounted filesystem. kept in inode = 0
         * and dev = devt of mounted filesystem in inode table. */
                struct mountnode
                {
                  struct inode  *_iplog;    /*itab of log*/
                  struct inode  *_ipinode;  /*itab of .inodes*/
                  struct inode  *_ipind;    /*itab of .indirect*/
                  struct inode  *_ipinomap; /*itab of inode map*/
                  struct inode  *_ipdmap;   /*itab of disk map*/
                  struct inode  *_ipsuper;  /*itab of super blk*/
                  struct inode  *_ipinodex; /*itab of .inodex*/
                }_mt_info;
```

```
/* offsets of MOUNT data */
#       define di_iplog _di_info._mt_info._iplog
#       define di_ipinode       _di_info._mt_info._ipinode
#       define   di_ipind       _di_info._mt_info._ipind
#       define di_ipinomap      _di_info._mt_info._ipinomap
#       define di_ipdmap        _di_info._mt_info._ipdmap
#       define di_ipsuper       _di_info._mt_info._ipsuper
#       define di_ipinodex      _di_info._mt_info._ipinodex
        /*
         * log info. kept in inode = 0 and dev = devt of log device
          filesystem in inode table.
         */
                struct lognode
                {
                int  _logptr   /* page number end of log */
                int  _logsize  /* log size in pages */
                int  _logend   /* eor in page _logptr */
                int  _logsync  /* addr in last syncpt record */
                int  _nextsync /* bytes to next logsyncpt */
                struct gnode * _logdgp; /* pointer to device gnode*/
                }_di_log;
/* offsets of LOG data */
#       define di_logptr        _di_info._di_log._logptr
#       define di_logsize       _di_info._di_log._logsize
#       define di_logend        _di_info._di_log._logend
#       define di_logsync       _di_info._di_log._logsync
#       define di_nextsync      _di_info._di_log._nextsync
#       define di_logdgp        _di_info._di_log._logdgp
        }_di_info;
};
```

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/include/sys/ino.h	The path to the **ino.h** header file, which defines the structure of an **inode** file.
/usr/include/sys/types.h	The path to the **types.h** header file.

Related Information

The **fs** file format, **stat** file format, **types.h** file format.

The File Systems Overview in *General Concepts and Procedures* explains file system types, management, structure, and maintenance.

The Directories Overview in *General Concepts and Procedures* explains working with directories and path names.

The Files Overview in *General Concepts and Procedures* provides information on working with files.

limits File

Purpose

Defines process resource limits for each user.

Description

The **/etc/security/limits** file is an ASCII file that contains stanzas with the process resource limits for each user. Each stanza is identified by a user name followed by a colon, and contains attributes in the *Attribute=Value* form. Each attribute is ended by a new line character, and each stanza is ended by an additional new line character. For an example of a stanza, see the Examples section.

If an attribute is not defined in the **/etc/security/limits** file, the values for the default user are applied. The attributes in the stanzas are as follows:

fsize The largest file a user's process can create or extend. The value is a decimal integer string.

core The largest core file a user's process can create. Must be in units of 512-byte blocks. The value is a decimal integer string. Not used.

cpu The largest amount of system unit time (in seconds) that a user's process can use. The value is a decimal integer string. Not used.

data The largest process data segment for a user's process. Must be in units of 512-byte blocks. The value is a decimal integer string. Not used.

stack The largest process stack segment for a user's process. Must be in units of 512-byte blocks. The value is a decimal integer string. Not used.

rss The largest amount of physical memory a user's process can allocate. Must be in units of 512-byte blocks. The value is a decimal integer string. Not used.

The **mkuser** command creates a user stanza in this file, using the attribute values defined in the **/etc/security/mkuser.default** file. An administrator can reset the attributes with the **chuser** command. To display the values, use the **lsuser** command. To remove a stanza, use the **rmuser** command.

Access to the user database files should be through the system commands and subroutines defined for this purpose. Access through other commands or subroutines may not be supported in future releases.

Security

Access Control: This file should grant read (r) access to the root user and members of the security group, and write (w) access only to the root user. Access for other users and groups depends upon the security policy for the system.

Auditing Events:

Event	Information
S_LIMITS_WRITE	filename

Example

A typical record looks like the following example for user `dhs`:

```
dhs:
        fsize = 8192
        core = 4096
        cpu = 3600
        data = 1024
        stack = 1024
        rss = 1024
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/limits	Specifies the path to the file.
/etc/group	Contains the basic group attributes.
/etc/security/group	Contains the extended attributes of groups.
/etc/passwd	Contains the basic user attributes.
/etc/security/passwd	Contains password information.
/etc/security/user	Contains the extended attributes of users.
/etc/security/environ	Contains the environment attributes of users.
/etc/security/audit/config	Contains audit system configuration information.
/etc/security/mkuser.default	Contains the default values for user accounts.

Related Information

The **chuser** command, **lsuser** command, **mkuser** command, **rmuser** command.

The **enduserdb** subroutine, **getuserattr** subroutine, **IDtouser** subroutine, **nextuser** subroutine, **putuserattr** subroutine, **setuserdb** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

login.cfg File

Purpose

Contains configuration information for log in and user authentication.

Description

The **/etc/security/login.cfg** file is an ASCII file with stanzas of configuration information for log in and user authentication. Each stanza has a name, followed by a : (colon), that defines its purpose. Attributes are in the form *Attribute*=*Value*. Each attribute is ended with a new line character, and each stanza is ended with an additional new line character. For an example of a stanza, see the Examples section.

There are three types of stanzas:

port	Defines the login characteristics of ports.
authentication method	Defines the authentication methods for users.
user configuration	Defines programs that change user attributes.

Each of these types of stanzas is described below.

Port Stanzas

Port stanzas define the login characteristics of ports and are named with the full path name of the port. Each stanza has the following attributes:

sak_enabled Defines whether the secure attention key (SAK) is enabled for the port. The SAK key is the Control–x, Control–r key sequence. Possible values for the **sak_enabled** attribute are:

- **true** SAK processing is enabled, so the key sequence establishes a trusted path for the port.
- **false** SAK processing is not enabled, so a trusted path cannot be established. This is the default value.

herald Defines the login message that is printed when the **getty** process opens the port. The default herald is the `login` prompt. The value is a character string.

synonym Defines other path names for the terminal. This attribute revokes access to the port and is used only for trusted-path processing. The path names should be device special files with the same major and minor number and should not include hard or symbolic links. The value is a list of comma-separated path names.

Authentication Method Stanzas

These stanzas define the authentication methods for users that are assigned in the **/etc/security/user** file. The name of each stanza must be identical to one of the methods defined by the **auth1** or the **auth2** attribute in the **/etc/security/user** file.

Each stanza has one attribute:

program Contains the full path name of a program that provides primary or secondary authentication for a user. Program flags and parameters may be included.

Since the SYSTEM authentication method is supported directly by the **login** command and the **su** command, and the NONE method does not provide any authentication, neither requires definition. However, all other authentication methods must be defined in this file. Different authentication methods can be defined for each user.

User Configuration

User configuration stanzas provide configuration information for programs that change user attributes. There are two stanzas of this type: pw_restrictions and usw.

The pw_restrictions stanza includes the restrictions for user-defined passwords that are applied by the **passwd** command and the **pwdadm** command. Valid restrictions follow:

maxage Defines the maximum age (in weeks) of a password. The password must be changed by this time. The value is a decimal integer string. The default is a value of 0, indicating no maximum age.

maxrepeats Defines the maximum number of times a character can be repeated in a new password. Since a value of 0 is meaningless, the PASS_MAX=8 value is used to indicate that there is no maximum number; this is the default value. The value is a decimal integer string.

minage Defines the minimum age (in weeks) a password must be before it can be changed. The value is a decimal integer string. The default is a value of 0, indicating no minimum age.

minalpha Defines the minimum number of alphabetic characters that must be in a new password. The value is a decimal integer string. The default is a value of 0, indicating no minimum number.

mindiff Defines the minimum number of characters required in a new password that must were not in the old password. The value is a decimal integer string. The default is a value of 0, indicating no minimum number.

minother Defines the minimum number of non-alphabetic characters that must be in a new password. The value is a decimal integer string. The default is a value of 0, indicating no minimum number.

The usw stanza defines the configuration of miscellaneous facilities. The following attributes can be included:

maxlogins Defines the maximum number of simultaneous local logins to the system. The format is a decimal integer string. The default is 0, indicating that there is no limit on simultaneous local login attempts.

shells Defines the valid shells on the system. This attribute is used by the **chsh** command to determine which shells a user can select. The value is a list of comma-separated full path names. The default is **/bin/sh**, **/bin/bsh**, **/bin/csh**, **/bin/ksh**.

Security

Access Control: This command should grant read (r) and write (w) access to the root user and members of the security group.

Auditing Events:

Event	Information
S_LOGIN_WRITE	filename

Examples

1. A typical `pw_restrictions` stanza looks like the following:

```
pw_restrictions:
  maxage = 12
  minother = 1
  minalpha = 4
```

   This example configures a maximum age of `12` weeks and a minimum of `1` non-alphabetic and `4` alphabetic characters for a new password.

2. A typical authentication_method stanza looks like the following:

```
meth1:
  program = /bin/auth_meth1
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/login.cfg	Specifies the path to the file.
/etc/group	Contains the basic attributes of groups.
/etc/security/group	Contains the extended attributes of groups.
/etc/passwd	Contains the basic attributes of users.
/etc/security/passwd	Contains password information.
/etc/security/user	Contains the extended attributes of users.
/etc/security/environ	Contains the environment attributes of users.
/etc/security/limits	Contains the process resource limits of users.
/etc/security/audit/config	Contains audit system configuration information.

Related Information

The **chfn** command, **chsh** command, **login** command, **passwd** command, **pwdadm** command, **su** command.

The **newpass** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures.*

mkuser.default File

Purpose

Contains the default attributes for new users.

Description

The **/etc/security/mkuser.default** file is an ASCII file that contains stanzas with attribute default values for users created by the **mkuser** command. Each attribute has the *Attribute*=*Value* form. If an attribute has a value of $USER, the **mkuser** command substitutes the name of the user. For a list of possible attributes, see the **chuser** command.

Each attribute is ended by a new line character, and each stanza is ended by an additional new line character.

There are two stanzas, user and admin, that can contain all defined attributes except the **auditclasses, id,** and **admin** attributes. The **mkuser** command generates a unique **id** attribute. The **admin** attribute depends on whether the **–a** flag is used with the **mkuser** command. The **auditclasses** attribute must be set explicitly by the root user or a member of the audit group.

For an example of a stanza, see the Example section.

Security

Access Control: If read (r) access is not granted to all users, members of the security group should be given read (r) access. This command should grant write (w) access only to the root user.

Example

A typical `user` stanza looks like the following:

```
user:
        pgrp = staff
        home = /u/$USER
        shell = /bin/sh
        auth1 = SYSTEM;$USER
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

File

/etc/security/mkuser.default	Specifies the path to the file.

Related Information

The **chuser** command, **mkuser** command.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

objects File

Purpose

Contains information about audited objects (files).

Description

The **/etc/security/audit/objects** file is an ASCII stanza file that contains information about audited objects (files). This file contains one stanza for each audited file. The stanza has a name equal to the path name of the file. Each file attribute has the following format:

access_mode = *audit_event*

An audit-event name can be up to 15 bytes long; longer names are rejected. Valid access modes are read (r), write (w), and execute (x) modes. For directories, search mode is substituted for execute mode.

Security

Access Control: This file should grant read (r) access to the root user and members of the audit group and grant write (w) access only to the root user.

Example

To define the audit events for the **/bin/passwd** file, add a stanza to the **objects** file:

```
/bin/passwd:
   x = PASSWD_Execute
   w = TCBAUTH_Modify
```

These attributes generate a PASSWD_Execute audit event each time the **bin/passwd** file executes and a TCBAUTH_Modify audit event each time the file is opened for writing.

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/audit/objects	Specifies the path to the file.
/etc/security/audit/config	Contains audit system configuration information.
/etc/security/audit/events	Contains the audit events of the system.
/etc/security/audit/bincmds	Contains auditbin backend commands.
/etc/security/audit/streamcmds	Contains auditstream commands.

Related Information

The **audit** command.

The **auditobj** subroutine.

To see the steps you must take to establish an Auditing System, refer to How to Set Up an Auditing System in *General Concepts and Procedures*. For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

/etc/passwd File

Purpose

Contains basic user attributes.

Description

The **/etc/passwd** file is an ASCII file that contains basic user attributes. The entry for each user has the following attributes, each separated by a colon:

Name — The login name of a user. The user name must be a unique alphanumeric string of 8 characters or less that begins with an alphabet character, and cannot be the **ALL** keyword or the **default** keyword. If there are duplicate login names in the file, the first entry is used.

Password — This field can contain a valid encrypted password, an * (asterisk) indicating an invalid password, or an ! (exclamation point) indicating that the password is in the **/etc/security/passwd** file. Under normal conditions, the field contains an !. If the field has an * and a password is required for user authentication, the user cannot log in.

UserID — The user's unique numeric ID that is used for discretionary access control. The value is a decimal numeric string.

PrincipleGroup — The user's principle group ID. This must be the numeric ID of a group in the user database or a group defined by a network information service. The value is a decimal numeric string.

Gecos — General information about the user that is not needed by the system, such as an office or phone number. The value is a character string.

HomeDirectory — The full path name of the user's home directory. If the user does not have a defined home directory, the home directory of the guest user is used. The value is a character string.

Shell — The initial program or shell that is executed after a user invokes the **login** command or **su** command. If a user does not have a defined shell, **/bin/sh,** the system shell, is used. The value is a character string that may contain arguments to pass to the initial program.

Note: Since the colon character is a field separator, it cannot be used in any attribute.

Users can have additional attributes in other system files. See the Files section for additional information.

Access to all of these user database files should be through the system commands and subroutines defined for this purpose. Access through other commands or subroutines may not be supported in future releases.

The **mkuser** command adds new entries to the **/etc/passwd** file and fills in the attribute values as defined in the **/etc/security/mkuser.default** file.

The *Password* attribute is always initialized to an * (asterisk), an invalid password. The password must be set with the **passwd** command or the **pwdadm** command. When the password is changed, an ! (exclamation point) is added to the **/etc/passwd** file, indicating that the encrypted password is in the **/etc/security/passwd** file.

Use the **chuser** command to change all user attributes except *Password*. The **chfn** command and the **chsh** command change the *Gecos* attribute and *Shell* attribute, respectively. To display all the attributes in this file, use the **lsuser** command. To remove a user and all the user's attributes, use the **rmuser** command. To write programs that affect attributes in the **/etc/passwd** file, use the subroutines listed in Related Information.

Security

Access Control: This file should grant read (r) access to all users and write (w) access only to the root user and members of the security group.

Examples

1. Typical records that show an invalid password for `smith` and `guest` follow:

   ```
   smith:*:100:100:8A-74(office):/u/smith:/bin/sh
   guest:*:200:0::/u/guest:/bin/sh
   ```

   The fields are in the following order: user name, password, user ID, primary group, general (gecos) information, home directory, and initial program (login shell). The * (asterisk) in the password field indicates that the password is invalid. For an explanation of each field, see the Description section.

2. If the password for `smith` in the previous example is changed to a valid password, the record will change to the following:

   ```
   smith:!:100:100:8A-74(office):/u/smith:/bin/sh
   ```

   The ! (exclamation point) indicates that an encrypted password is stored in the **/etc/security/passwd** file.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/passwd	Specifies the path to the file.
/etc/group	Contains the basic attributes of groups.
/etc/security/group	Contains the extended attributes of groups.
/etc/security/passwd	Contains password information.
/etc/security/user	Contains the extended attributes of users.
/etc/security/environ	Contains the environment attributes of users.
/etc/security/limits	Contains the process resource limits of users.

Related Information

The **chfn** command, **chsh** command, the **chuser** command, the **lsuser** command, **mkuser** command, **passwd** command, **pwdadm** command, **rmuser** command.

The **endpwent** subroutine, **enduserdb** subroutine, **getpwent** subroutine, **getpwnam** subroutine, **getpwuid** subroutine, **getuserattr** subroutine, **IDtouser** subroutine, **nextuser** subroutine, **putpwent** subroutine, **putuserattr** subroutine, **setuserdb** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

/etc/security/passwd File

Purpose

Contains password information.

Description

The **/etc/security/passwd** file is an ASCII file that contains stanzas with password information. Each stanza is identified by a user name followed by a : (colon) and contains attributes in the form *Attribute=Value*. Each attribute is ended with a new line character, and each stanza is ended with an additional new line characher. For a typical stanza, see the Examples section below.

Each stanza can have the following attributes:

passwd The encrypted password. The system encrypts the password created with the **passwd** command or the **pwdadm** command. If the password is NULL or empty, the user does not have a password. If the password is an * (askerisk), the user cannot log in. The value is a character string. The default value is *.

lastupdate The time (in seconds) since the epoch (00:00:00 GMT, January 1, 1970) when the password was last changed. If password aging (the **minage** attribute or the **maxage** attribute) is in effect, the **lastupdate** attribute forces a password change when the time limit expires. (See the **/etc/security/login.cfg** file for information on password aging.) The **passwd** command and the **pwdadm** command normally set this attribute when a password is changed. The value is a decimal integer.

flags The restrictions applied by the **login** command and the **su** command. The value is a list of comma–separated attributes. The **flags** attribute can be be left blank or can be one or more of the following:

- **ADMIN** Defines the administrative status of the password information. If the **ADMIN** attribute is set, only the root user can change this password information.
- **ADMCHG** Indicates that the password was last changed by a member of the security group. Normally this flag is set only implicitly when the **pwdadm** command or the **passwd** command change another user's password. When this flag is set explicitly, it forces the password to be updated the next time a user gives the **login** command or the **su** command.
- **NO_CHECK** None of the system password restrictions defined in the **/etc/security/login.cfg** file are enforced for this password.

When the **passwd** command or the **pwdadm** command updates a password, the command adds values for the **passwd** and **lastupdate** attributes and, if used to change another user's password, for the **flags ADMCHG** attribute.

Access to this file should be through the system commands and subroutines defined for this purpose. Other accesses may not be supported in future releases. Users can update their own passwords with the **passwd** command, administrators can set passwords and password flags with the **pwdadm** command, and the root user is able to use the **passwd**

command to set the passwords of other users. For information on access through programs, see the subroutines in the Related Information section below.

Refer to the Files section below for information on where attributes and other information on users and groups are stored.

Although each user name must be in the **/etc/passwd** file, it is not necessary to have each user name listed in the **/etc/security/passwd** file. If the authentication attributes **AUTH1** and **AUTH2** are so defined in the **/etc/security/user** file, a user may use the authentication name of another user. For example, the authentication attributes for user tom can allow him to use the entry in the **/etc/security/passwd** file for user carol for authentication.

Security

Access Control: This file should grant read (r) and write (w) access only to the root user.

Auditing Events:

Event	Information
S_PASSWD_READ	filename
S_PASSWD_WRITE	filename

Examples

The following line indicates that the password information in the **/etc/security/passwd** file is available only to the root user, who has no restrictions on updating a password for the specified user:

```
flags = ADMIN,NOCHECK
```

An example of this line in a typical stanza for user `smith` follows:

```
smith:
  passwd = MGURSj.F056Dj
  lastupdate = 623078865
  flags = ADMIN,NOCHECK
```

The `passwd` line shows an encrypted password. The `lastupdate` line shows the number of seconds since the epoch that the password was last changed. The `flags` line shows two flags; the ADMIN flag indicates that the information is available only to the root user, and the NOCHECK flag indicates that the root user has no restrictions on updating a password for the specified user.

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

File

/etc/security/passwd Specifies the path to the file.

Related Information

The **login** command, **ftpd** command, **passwd** command, **pwdadm** command, **rlogind** command, **su** command.

The **endpwdb** subroutine, **getuserpw** subroutine, **putuserpw** subroutine, **setpwdb** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures.*

PC Simulator ttylog File

Purpose

Serves as an aid in identifying problems in the interaction (keystrokes and display data) between PC Simulator and an application program.

Note: This file can be used only on a terminal running in monochrome mode, whether the terminal is ASCII or not.

Description

The **ttylog** file is generated by adding the TRACE option to an ASCII terminal configuration file before running the application. The **ttylog** file is a log of the keystroke input and display data passed by PC Simulator to the application program.

Example

The example shown here is a partial **ttylog** file processed with the **od** command.

The left-most column gives the byte offset into the file, expressed as an octal number. The rest of each numbered row consists of groups of four hexadecimal digits representing either display characters or characters entered from the keyboard. Display characters are expressed as their ASCII character codes. Keystrokes are represented by their scan codes. Immediately below each numbered row is a row of printable ASCII characters. Most of these are characters displayed on the screen.

Asterisks represent repeating rows of blank spaces (hexadecimal 20).

```
Offset
(octal) Synchronous display—output/keyboard—input

0000000  ff43  7572  7265  6e74  2064  6174  6520  6973
         C  u  r  r  e  n  t     d  a  t  e     i  s
0000020  2054  6875  2031  312d  3033  2d31  3938  3820
         T  h  u     1  1  -  0  3  -  1  9  8  8
0000040  2020  2020  2020  2020  2020  2020  2020  2020
*
0000120  2045  6e74  6572  206e  6577  2064  6174  6520
         E  n  t  e  r     n  e  w     d  a  t  e
0000140  286d  6d2d  6464  2d79  7929  3a20  2020  2020
         (  m  m  -  d  d  -  y  y  )  :
0000160  2020  2020  2020  2020  2020  2020  2020  2020
*
0000240  2043  7572  7265  6e74  2074  696d  6520  6973
         C  u  r  r  e  n  t     t  i  m  e     i  s
0000260  2031  343a  3433  3a21  382e  3337  2020  2020
         1  4  :  4  3  :  1  8  .  3  7
0000300  2020  2020  2020  2020  2020  2020  2020  2020
*
0000360  2045  6e74  6572  206e  6577  2074  696d  653a
         E  n  t  e  r     n  e  w     t  i  m  e  :
0000400  2020  202—0  2020  2020  2020  2020  2020  2020
*
0000740  2054  6865  2049  424d  2050  6572  736f  6e61
         T  h  e     I  B  M     P  e  r  s  o  n  a
0000760  6c20  436f  6d70  7574  6572  2044  4f53  2020
         l     C  o  m  p  u  t  e  r     D  O  S
```

PC Simulator ttylog

```
0001000  2020  2020  2020  2020  2020  2020  2020  2020
*
0001060  2056  6572  7369  6f6e  2033  2e33  3020  2843
        V  e  r  s  i  o  n     3  .  3  0     (  C
0001100  2943  6f70  7972  6967  6874  2049  6e74  6572
        )  C  o  p  y  r  i  g  h  t     I  n  t  e  r
0001120  6e61  7469  6f6e  616c  2042  7573  696e  6573
        n  a  t  i  o  n  a  l     B  u  s  i  n  e  s
0001140  7320  4d61  6368  696e  6573  2043  6f72  7020
        s     M  a  c  h  i  n  e  s     C  o  r  p
0001160  3139  3831  2c20  3139  3837  2020  2020  2020
        1  9  8  1  ,  1  9  8  7
0001200  2020  2020  2020  2020  2020  2020  2020  2843
        (  C
0001220  2943  6f70  7972  6967  6874  204d  6963  726f
        )  C  o  p  y  r  i  g  h  t     M  i  c  r  o
0001240  736f  6674  2043  6f72  7020  3139  3831  2c20
        s  o  f  t     C  o  r  p     1  9  8  1  ,
0001260  3139  3836  2020  2020  2020  2020  2020  2020
        1  9  8  6
0001300  2020  2020  2020  2020  2020  2020  2020  2020
*
0001440  2043  3e20  2020  2020  2020  2020  2020  2020
        C  >
0001460  2020  2020  2020  2020  2020  2020  2020  2020
*
0003600  20ff  2eae  ff63  20ff  20a0  ff63  64ff  39b9
                       c                    c  d     9
0003620  2bab  ff20  5cff  1f9f  ff73  20ff  25a5  ff73
        +          \   \s1 237    s       %       s
0003640  6bff  1c9c  ff43  3eff  20a0  ff64  20ff  1797
        k   \s4       C  >            d      027
0003660  ff64  69ff  1393  ff72  20ff  1c9c  ff20  566f
        d  i   023  o     r      \s4          V  o
0003700  6c75  6d65  2069  6e20  6472  6976  6520  4320
        l  u  m  e     i  n     d  r  i  v  e     C
0003720  6861  7320  6e6f  206c  6162  656c  2020  4469
0003720  6861  7320  6e6f  206c  6162  656c  2020  4469
        j  a  s     n  o     l  a  b  e  l        D  i
0003740  7265  6374  6f72  7920  6f66  2020  433a  5c53
        r  e  c  t  o  r  y     o  f        C  :  \  S
```

Using the ttylog File

If you suspect that problems in executing your program on an ASCII terminal are a result of the keystrokes being passed by PC Simulator to the application program, use the TRACE option and the **ttylog** file to determine exactly what keystrokes the program is receiving.

A few application programs handle keyboard input in a way that makes them vulnerable to problems caused by differences between an ASCII terminal and a IBM Personal Computer. The differences lie in the handling of the characters ESC, TAB, ENTER (Return), and BACK (Backspace). These characters can be generated on the terminal by pressing a key with the proper label *or* by pressing CTRL–[, CTRL–i, CTRL–m, and CTRL–h, respectively.

On the IBM Personal Computer, however, while CTRL–h and Backspace both move the cursor back one character, each generates a different scan code. (Scan codes sent by PC Simulator to the IBM Personal Computer are listed in PC Key Names and Corresponding Make Scan Codes.) If your DOS application expects a Backspace character, pressing either the Backspace key or the CTRL–h terminal keys generates the expected character. But If the application really expects CTRL–h, you must define a terminal key sequence that translates to CTRL + h in the #ACTION stanza. This generates the correct scan code for the DOS application.

The example log file shown above records a simulator session in which the following sequence of events occurs:

1. DOS starts up.
2. DOS displays the date prompt.
3. The user presses the Enter key, accepting the date shown.
4. DOS displays the time prompt.
5. The user presses the Enter key, accepting the time shown.
6. DOS displays the system prompt: **C>**.
7. The user enters the **CD** command: **CD \sk**.
8. The user presses the Enter key.
9. The user enters the **DIR** command: **DIR**.
10. The user presses the Enter key.
11. The user enters the **DIR** command: **DIR**.
12. DOS displays the following messages:

```
Volume in drive C has no label

Directory of C:\SK

(directory listing not shown)
```

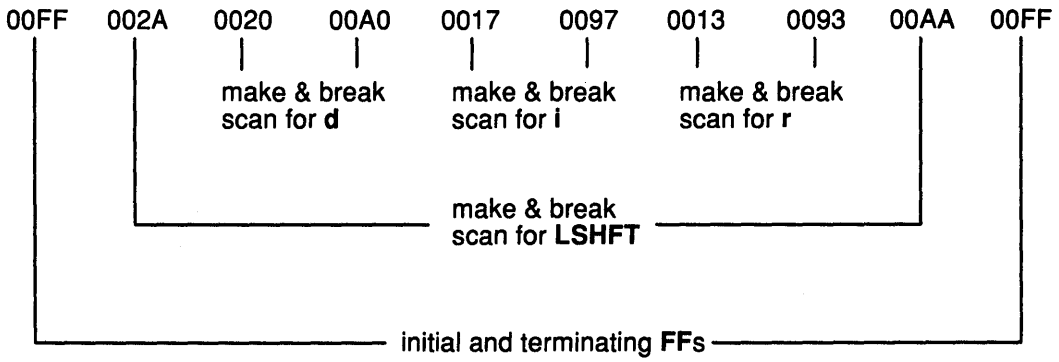
Format of the ttylog File

When using the **ttylog** file to solve a problem in the interaction between a simulator session on the ASCII terminal and a DOS application, you are usually concerned with the keystrokes being sent to the application rather than the display characters. PC Simulator encodes keyboard scan–code sequences sent to the DOS application as follows:

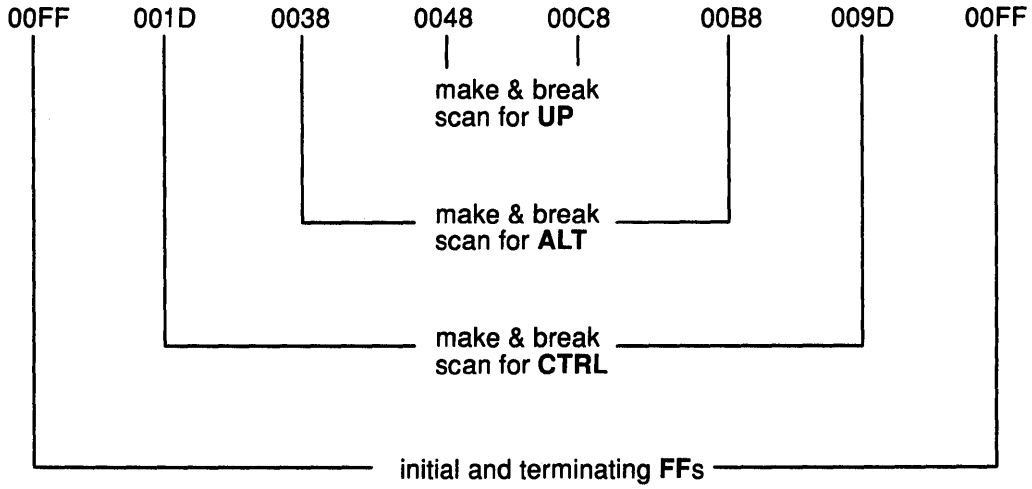
```
FF 2EAE FF
```

The format of the sequence is: an initial hexadecimal FF, the make scan code for the primary keystroke(s), the break scan code for the primary keystroke(s), and a terminating FF. PC Key Names and Corresponding Make Scan Codes lists the make scan codes for the IBM Personal Computer keys. In the example just shown, 2E is the make scan code for the character C. You form the break scan code for any character by turning on the most significant bit of its make scan code. You can also think of it as adding 0x80 to the make scan code (for example, AE is the break scan code for the character C). A terminating hexadecimal FF follows the break scan code.

The addition of a "state" key—such as CTRL, LSHFT, RSHFT, and ALT—changes the encoding format slightly. For example, the following diagram shows how PC Simulator encodes the **DIR** command entered in uppercase (shifted):



PC Simulator brackets the makes and breaks of the primary key(s) between the makes and breaks of any state keys. The same encoding format is used when multiple state keys are sent to a DOS application, an in the following example of CTRL–ALT–UP (cursor–up).

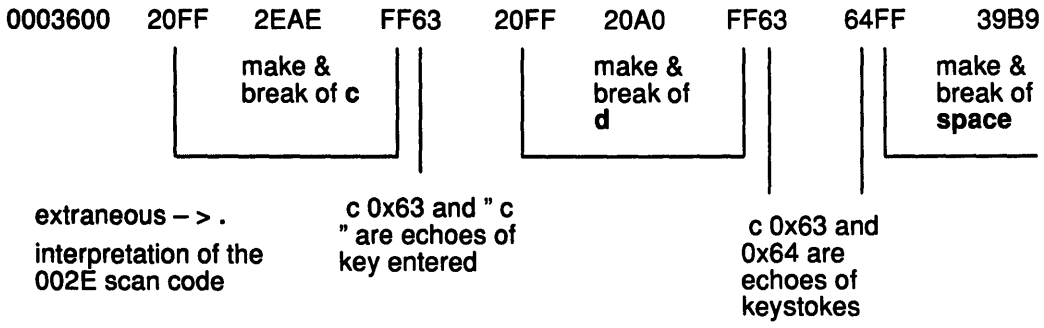


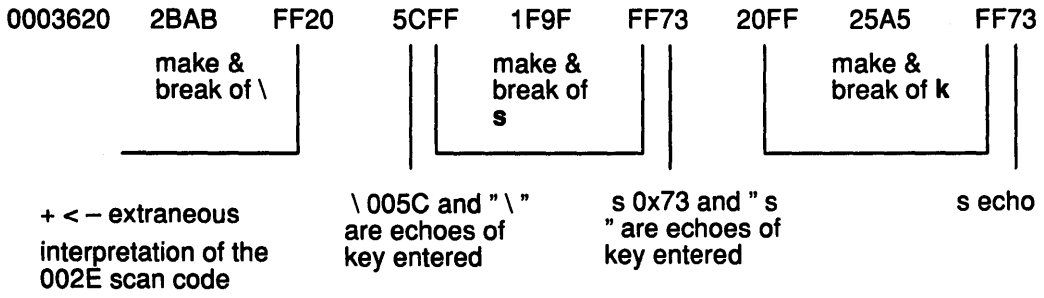
There are two aspects of the data in **ttylog** files that appear anomalous:

1. Because the AIX **od** command treats any two adjacent hexadecimal digits as a printable character, extraneous characters may appear in the printable character line. They are not entered from the keyboard or displayed on the screen.
2. The **ttylog** file often contains bytes that do not seem to fit the flow of events. A common example is two sets of hexadecimal FFs with no digits between them. These are not keystroke delimiters. PC Simulator produces them as a response to an internal event and they can be ignored. The file may also contain extra "blank" (0x20) characters. The explanation for their presence is that PC Simulator refreshes the terminal display in groups of four characters. If you type a character on a blank line, PC Simulator echoes the character on the display together with three "blank" characters following it.

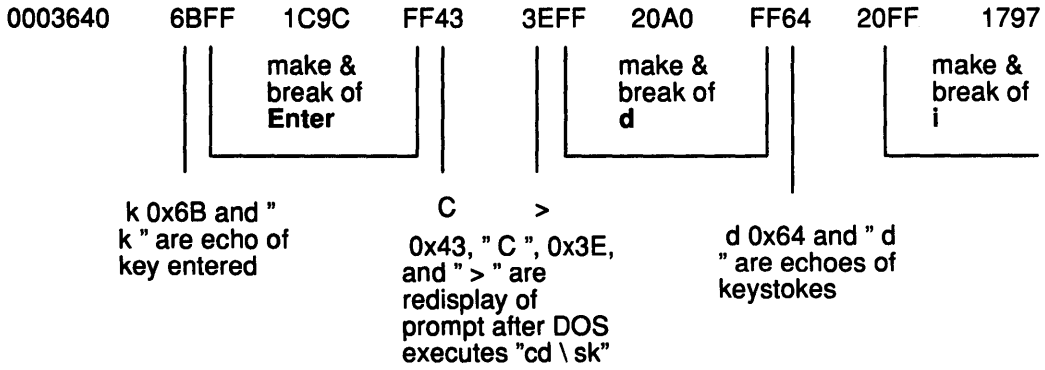
Decoding PC Simulator Events

At offset 1440 in the example **ttylog** file, DOS displays the **C>** prompt, and the remainder of the screen is filled with blanks. This accounts for the many missing rows, designated by asterisks. The first keyboard input is at offset 3600 in the file. The keys encoded there are the representation of event 7 in the session. The entry **CD \sk** is decoded as shown in the following diagram. Large "brackets" have been added to make the data more readable.





At offsett 0x3540 DOS has executed the change directory command and event 9 begins as shown in the following example.



PC Key Names and Corresponding Make Scan Codes

PC Key Name	Hex Scan-Code	PC Key Name	Hex Scan-code
ESC	01	LSHFT	2A
1	02	BSLSH	2B
2	03	Z	2C
3	04	X	2D
4	05	C	2E
5	06	V	2F
6	07	B	30
7	08	N	31
8	09	M	32
9	0A	COMMA	33
0	0B	DOT	34
DASH	0C	SLASH	35
EQUAL	0D	RSHFT	36
BACK	0E	ASTR	37
TAB	0F	ALT	38
Q	10	SPACE	39
W	11	CAPS	3A
E	12	F1	3B
R	13	F2	3C
T	14	F3	3D
Y	15	F4	3E
U	16	F5	3F
I	17	F6	40
O	18	F7	41
P	19	F8	42
LBRAK	1A	F9	43
RBRAK	1B	F10	44
ENTER	1C	NUM	45
CTRL	1D	SCROL	46
A	1E	HOME	47
S	1F	UP	48
D	20	PGUP	49
F	21	MINUS	4A
G	22	LEFT	4B
H	23	N5	4C
J	24	RIGHT	4D
K	25	PLUS	4E
L	26	END	4F
SEMIC	27	DOWN	50
QUOTE	28	PGDOWN	51
ACENT	29	INS	52
		DEL	53

How to Generate a ttylog

1. PC Simulator must be running in PC monochrome mode.
2. Add the TRACE option to the terminal configuration file (in the #DEFINE stanza), using the following format:

   TRACE=*nonzero_value*

   Note: The TRACE option must be uppercase, as shown. Also make sure that no comment designator (;) precedes it on the line.

   This option creates a log file of the keystrokes sent to the application being run on PC Simulator. The file is called **/usr/lpp/pcsim/tty/ttylog**.

3. To examine the contents of the log file, enter the AIX **od** command, using the following format:

   ```
   od –cx /usr/lpp/pcsim/tty/ttylog > file–name
   ```

   Issuing this command at the AIX prompt writes the **ttylog** contents into *file–name* in hexadecimal format with an ASCII character equivalent for each byte.

   Note: The *file–name* file should be inspected on a IBM RISC System/6000 display screen rather than your ASCII terminal display screen, because the file may contain characters that are interpreted as line–control characters by an ASCII terminal.

Files

/usr/lpp/pcsim/tty/ascii	Default terminal configuration file.
/usr/lpp/pcsim/tty/ibm3151–11	Terminal configuration file for the IBM 3151 terminal.
/usr/lpp/pcsim/tty/ibm3151–31	Terminal configuration file for the IBM 3151 terminal.
/usr/lpp/pcsim/tty/ibm3151–51	Terminal configuration file for the IBM 3151 terminal.
/usr/lpp/pcsim/tty/ibm3161	Terminal configuration file for the IBM 3161 terminal.
/usr/lpp/pcsim/tty/ibm3162	Terminal configuration file for the IBM 3162 terminal.
/usr/lpp/pcsim/tty/ibm3163	Terminal configuration file for the IBM 3163 terminal.
/usr/lpp/pcsim/tty/ibm3164	Terminal configuration file for the IBM 3164 terminal.
/usr/lpp/pcsim/tty/ibmhft	Terminal configuration file for PC monochrome mode on the IBM AIX local display.
/usr/lpp/pcsim/tty/ibmxterm	Terminal configuration file for PC monochrome mode on the IBM AIXwindows display.
/usr/lpp/pcsim/tty/keyboard.sys	Terminal configuration file for world trade 102–key keyboard for UK, France, Italy, Sweden in AIXwindows.
/usr/lpp/pcsim/tty/ttyxlat	Keyboard mapping file.
/usr/lpp/pcsim/tty/vt100	Terminal configuration file for the DEC VT100 terminal.
/usr/lpp/pcsim/tty/vt220	Terminal configuration file for the DEC VT220 terminal.
/usr/lpp/pcsim/tty/vt320	Terminal configuration file for the DEC VT320 terminal.
/usr/lpp/pcsim/tty/vt330	Terminal configuration file for the DEC VT330 terminal.
/usr/lpp/pcsim/tty/wy30	Terminal configuration file for the Wyse WY30 terminal.
/usr/lpp/pcsim/tty/wy50	Terminal configuration file for the Wyse WY50 terminal.
/usr/lpp/pcsim/tty/wy60	Terminal configuration file for the Wyse WY60 terminal.
/usr/lpp/pcsim/tty/wy350	Terminal configuration file for the Wyse WY350 terminal.

Related Information

PC Simulator Overview and PC Key Names and Corresponding Make Scan Codesin *General Concepts and Procedures*.

The **od** command and the **pcsim** command.

plot File

Purpose

Provides the graphics interface.

Description

The **plot** subroutines in the following list produce output files in the format outlined in this section. The **tplot** commands then interpret these graphics files for various devices, performing the plotting instructions in the order in which 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 4 bytes representing the x and y values: Each value is a 2-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 lists each **plot** instruction with its corresponding **plot** subroutines:

Instr	Subroutine Description
a **arc**	Draws the arc described by the following 12 bytes. The first 4 bytes describe the center point (x, y) of the arc or circle. The second 4 bytes describe the beginning point of the arc. The third 4 bytes describe the ending point of the arc. Arcs are drawn counterclockwise. 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 4 bytes, and whose radius is given as an integer in the following 2 bytes.
e **erase**	Starts another frame of output.
f **linemod**	Uses the string that follows it, 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 4 bytes to the point designated by the following 4 bytes.
m **move**	The next 4 bytes designate a new current point.
n **cont**	Draws a line from the current point to the point designated by the next 4 bytes.
p **point**	Plots the point designated by the next 4 bytes.
s **space**	The next 4 bytes designate the lower left corner of the plotting area, followed by 4 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 following space setting is as follows:

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

This 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.

File

TERM

Related Information

The **plot** subroutine.

The **graph** command and **tplot** command.

qconfig File

Purpose

Configures a printer queueing system.

Description

The **qconfig** file describes the queues and devices available for use by the **enq** command, which places requests on a queue, and the **qdaemon** command, which removes requests from the queue and processes them. The **qconfig** file is an attribute file.

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. Unless the **PRINTER** environment variable is set, the **enq** command uses this queue when it receives no queue parameter.

The name of a queue stanza must be 1 to 7 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.

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

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

up Defines the state of the queue. **TRUE**, the default, indicates that it is running. **FALSE** indicates that it is not running.

The following list shows some of the field names along with some of the possible values that appear in the **qconfig** file for remote queues:

host Remote host where the remote queue is found.

s_statfilter Short version filter used to translate remote queue status format. The default, **/usr/lpd/aixshort**, indicates that the remote print server is an IBM AIX Version 3 for RISC System/6000 machine and status information will be represented in short format. Other choices are **/usr/lpd/bsdshort** for BSD remote and **/usr/lpd/aixv2short** for the RT system.

l_statfilter Long version filter used to translate remote queue status format. The default, **/usr/lpd/aixlong**, indicates that the remote print server is an AIX for RISC System/6000 machine and status information will be represented in long format. Other choices are **/usr/lpd/bsdlong** for BSD remote, and **/usr/lpd/aixv2long** for the RT system.

rq Remote queue name.

If a field is omitted, its default value is assumed. The default values for a queue stanza are:

```
discipline      =       fcfs
up              =       TRUE
acctfile        =       FALSE
```

The device field cannot be omitted.

The name of a device stanza is arbitrary and can be 1 to 7 characters long. The fields that can appear in the 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 **TRUE**.

backend Specifies the full path name of the backend, optionally followed by flags and parameters to be passed to it. The path names most commonly used are **/usr/lpd/piobe** for local print and **/usr/lpd/rembak** for remote print.

feed Specifies the number of separator pages to print when the device becomes idle or the value **never**, the default, 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 and that the file name is **/dev/null**. 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** program 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   =       TRUE
```

The **enq** 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 the **/etc/qconfig.bin** file.

Examples

1. The batch queue supplied with the AIX for RISC System/6000 might contain these stanzas:

```
bsh:
        discipline = fcfs
        device = bshdev
bshdev:
        backend = /bin/ksh
```

   To run a shell procedure called `myproc` using this batch queue, enter:

```
qprt -Pbsh 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, enter:

```
bsh:
        discipline = fcfs
        device = bsh1,bsh2
bsh1:
        backend = /bin/ksh
bsh2:
        backend = /bin/ksh
```

3. To set up a remote queue `bsh`, enter:

```
remh:
        device = rd0
        host = pluto
        rq = bsh
rd0:
        backend = /usr/lpd/rembak
```

Files

/etc/qconfig	Configuration file.
/etc/qconfig.bin	Digested, binary version of the **/etc/qconig** file.
/usr/lpd/digest	Program that converts the **qconfig** file to binary.

Related Information

The **lp** command, **qdaemon** command, **enq** command.

streamcmds File

Purpose

Contains auditstream commands.

Description

The **/etc/security/audit/streamcmds** file is an ASCII template file that contains the stream mode commands that are invoked when the audit system is initialized. The path name of this file is defined in the stream stanza of the **/etc/security/audit/config** file.

This file contains command lines, each of which is composed of one or more commands with input and output that may be piped together or redirected. Although the commands usually are one or more of the audit system commands (the **auditcat** command, the **auditpr** command, the **auditselect** command), this is not a requirement. The first command, however, should be the **auditstream** command.

When the audit system is initialized, the **audit start** command runs each command. No path name substitution is performed on **$trail** or **$bin** strings in the commands.

Security

Access Control: This file should grant read (r) access to the root user and members of the audit group and grant write (w) access only to the root user.

Examples

1. To read all records from the audit device, select and format those that involve unsuccessful events, and print them on a line printer, include the following in the **/etc/security/audit/streamcmds** file:

   ```
   /etc/auditstream | /etc/auditselect -e \
    "result = fail" |/etc/auditpr -v > /dev/lpr0
   ```

   This command is useful for creating a hard copy trail of system security violations.

2. To read all records from the audit device that have audit events in the `authentication` class, format them, and display them on the system console, include the following in the **/etc/security/audit/streamcmds** file:

   ```
   /etc/auditstream -c authentication | \
   /etc/auditpr -t0 -v > /dev/console
   ```

   This command allows timely auditing of user authentication events.

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/audit/streamcmds	Specifies the path to the file.
/etc/security/audit/config	Contains audit system configuration information.
/etc/security/audit/events	Contains the audit events of the system.
/etc/security/audit/objects	Contains information about audited objects (files).
/etc/security/audit/bincmds	Contains auditbin backend commands.

Related Information

The **audit** command, **auditcat** command, **auditpr** command, **auditselect** command.

To see the steps you must take to establish an Auditing System, refer to How to Set Up an Auditing System in *General Concepts and Procedures*.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

sysck.cfg File

Purpose

Contains file definitions for the trusted computing base

Description

The **/etc/security/sysck.cfg** file is a stanza file that contains definitions of file attributes for the trusted computing base. The name of each stanza is the pathname of a file, followed by a **:** (colon). Attributes are in the form *Attribute* = *Value*. Each attribute is ended with a new line character, and each stanza is ended with an additional new line character. For an example of a stanza, see the Examples section below.

Each stanza can have one or more of the following attributes, and must have the **type** attribute:

acl Defines the access control list of the file, including the SUID, SGID, and SVTX bits. The value is the *Access Control List*, in the format described in Access Control Lists.

class Defines a group of files for checking, deleting, or updating. A file can be in more than one class. The value is the *ClassName* [*ClassName*]parameter.

checksum Defines the checksum, as computed with the **sysck checksum** program. This attribute is valid only for regular files. The value is the output of the **sum –r** command, including spaces.

group Defines the group name or numeric group ID, expressed as the *GroupName* or *GroupID* parameter.

links Defines the absolute paths that have hard links to this object. The value must be an absolute pathname, expressed as the *Path*, [*Path* ...] parameter.

mode Defines the file mode, expressed as *Flag*, *Flag* ..., *PBits*. The *Flag* parameter can contain the **SUID**, **SGID**, **SVTX**, and **tcb** mode attributes. The *PBITS* parameter contains the base file permissions, expressed either in octal form, such as 640, or symbolic form, such as rw–,r—, r—. The order of the attributes in the *Flag* parameter is not important, but base permissions must be the last entry in the list. The symbolic form may include only read (r), write (w), and execute (x) access. If the **acl** attribute is defined in the stanza, the **SUID**, **SGID**, and **SVTX** mode attributes are ignored. For a typical mode specification, see the Examples section below.

owner Defines the name or numeric ID of the file owner, expressed as the *OwnerName* or the *OwnerID* parameter.

program Defines an associated checking program. When the **sysck** command is in check mode (given with the **–n**, **–p**, **–t** or **–y** flag), the associated program is invoked as part of the check, with the flag as its first argument. The value must be an absolute pathname. If flags are specified, the value should be expressed as *Path* [,*Flag*].

size Defines the size of the file in bytes. This attribute is valid only for regular files. The value is a decimal number.

source Defines the source file. In check mode, the source file is copied to the file being checked to initialize it. If this attribute is blank or empty (**source =**) an empty file (regular file, directory, or named pipe*)* is created. This attribute requires a value if the object is a device file. The value must be an absolute pathname.

symlinks Defines the absolute paths that have symbolic links to this object. The value is *Path*, [*Path* ...]

type The type of object. Select one of the following: the **FILE**, **DIRECTORY**, **FIFO**, **BLK_DEV**, **CHAR_DEV**, or **MPX_DEV** keyword.

Stanzas in this file can be created and altered with the **sysck** command. Direct alteration by other means should be avoided, since other accesses may not be supported in future releases.

Attributes that span multiple lines must be enclosed in double quotes and have new line characters entered as **\n**.

Security

Access Control: This file should grant read (r) access to the root user and members of the security group, and write (w) access only to the root user. General users do not need read (r) access.

Examples

1. A typical stanza looks like the following example for the `/etc/passwd` file:

```
/etc/passwd:
        type = file
        owner = root
        group = passwd
        mode = TCB,640
        program = /bin/pwdck ,ALL
```

2. A typical mode specification looks like the following example for a program that is part of the Trusted Computing Base, is a Trusted Process, and which has the setuid attribute enabled:

```
mode = SUID,TP,TCB,rwxr-x---
```
Or
```
mode = SUID,TP,TCB,750
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/sysck.cfg Specifies the path to the system configuration data base.

Related Information

The **grpck** command, **installp** command, **pwdck** command, **sum** command, **sysck** command, **updatep** command, **usrck** command.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures.*

terminfo File

Purpose

Describes terminal 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 article explains the **terminfo** source file format. Before a **terminfo** source file can be used, it must be compiled using the **tic** command. You can edit and modify these source files, such as the **/usr/lib/terminfo/ibm.ti** file, which describes IBM terminals, and the **/usr/lib/terminfo/dec.ti** file, which describes DEC terminals.

See **TERM** for a list of some 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 *Terminal*–**w**. The following suffixes should be used where possible:

Suffix	Meaning	Example
–am	With automatic margins (usually default)	*Terminal*–**am**
–m	Monochrome mode	*Terminal*–**m**
–w	Wide mode (more than 80 columns)	*Terminal*–**w**
–nam	Without automatic margins	*Terminal*–**nam**
–n	Number of lines on the screen	*Terminal*–**60**
–na	No arrow keys (leave them in local)	*Terminal*–**na**
***–n*p**	Number of pages of memory	*Terminal*–**4p**
–rv	Reverse video	*Terminal*–**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 in lines beginning with the # (pound sign) character.

List of Capabilities

The following listing 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 data base. 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 **caps** source file 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 can be specified.

(G) Indicates that the string is passed through **tparm** with parameters as given (*#i*).

(*) Indicates that padding can be based on the number of lines affected.

(*#i*) Indicates the *ith* parameter.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
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	shp	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.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
teleray_glitch	xt	xt	Indicates destructive tabs and blanks inserted while entering standout mode.
tilde_glitch	hz	hz	Indicates terminal cannot print ~ (tilde) characters.
transparent_underline	ul	ul	Overstrikes with underline character.
xon_xoff	xon	xo	Indicates terminal uses xon/xoff handshaking.

Numbers

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
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 (zero) indicates variable.
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	Byx	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.
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	ed	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.

Strings (continued)

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
color_fg_0	colf0	0	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.
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	cub1	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	cuf1	nd	Indicates nondestructive space (cursor right).
cursor_to_ll	ll	ll	Moves cursor to first column of last line (if no **cup**).
cursor_up	cuu1	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_bold_mode	blink	mb	Enables blinking.
enter_blink_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_reserse_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.
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.

Strings (continued)

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
flash_screen	flash	vb	Indicates visual 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	ke1	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.
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.

Strings (continued)

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
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_new-line	knl	kn	New-line key.
key_next_pane	knpn	kv	Next-pane key.
key_n_page	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 key.
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	abels 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 F08
lab_f9	f9	l9	Labels function key F9 if not F9.
lab_f10	lf10	la	Labels function key F10 if not F10.

Strings (continued)

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
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.
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.

Strings (continued)

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
key_a1	ka1	K1	Specifies upper left of keypad.
key_a3	ka3	K3	pecifies 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 **am** capability. Hence the description of the terminal includes the **am** capability. Numeric capabilities are followed by the # (pound 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 the **el** capability (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 the **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 a 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 escape sequences include **\^** (backslash caret) for a ^ (caret), \ \ (backslash backslash) for a \ (backslash), \ , (backslash comma) for a, (comma), \ : (backslash colon) for a : (colon), and **\0** (backslash zero) for the null character. (**\0** will produce **\200**, which does not end 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 terminal 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 the **vi** editor 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 the **vi** editor. 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 the **/usr/lib/terminfo** file. A test to get the correct padding (if known) is to edit the **/etc/passwd** file at 9600 baud, delete about 16 lines from the middle of the screen, then press 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.

cuf1, **cuu1**, and **cud1**
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. The **os** capability applies to storage scope, printing, and APL terminals.

The **terminfo** file's initialization subroutine, **setupterm**, calls the **termdef** command to determine the number of lines and columns on the display. If the **termdef** command cannot supply this information, then the **setupterm** subroutine uses the **lines** and **cols** values in the data base.

A point to note here is that the local cursor motions encoded in the terminfo file are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless the **bw** string 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 **ind** string and the **ri** string 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 **cuf1** from the last column. The only local motion that is defined from the left edge is if the **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If the **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 the **am** capability. If the terminal has a command that moves to the first column of the next line, that command can be given as the **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, cud1=^J, hc, ind=^J, os,
```

And another terminal is described as:

```
xxxx | x | xxxxxxxx,
        am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^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 escape sequences 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 memeory relative cursor addressing, that can be indicated by the **mrcup** capability.

The parameterized capabilities and their descriptions are:

cub1 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.

hpa and **vpa** Indicates the cursor has row or column absolute cursor addressing, horizontal position absolute (**hpa**) and vertical 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 **ind** capability, and **ri** capability. *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 the **printf** command (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 the **printf** command (2 digits output).

%03d Prints as in the **printf** command (3 digits output).

%c Print pop() give **%c** (character output from stack).

%s Print pop() gives **%s** (string output from stack).

%p[*i*] Pushes the *ith* parameter onto the stack.

%P[*a-z*] Sets variable [*a-z*] to pop() (variable output 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:

```
%? c1 %t b1 %e c2 %t b2 %e c3 %t b3 %e b4 %;
```

In this example, the **c*i*** denotes conditions, and the **b*i*** denotes bodies.

Binary operations are in postifx 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%zdc%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 the **terminfo** file 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 the very upper left corner of the 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 position 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 is 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 is a parameterized string with 4 parameters: the starting and ending lines in memory and the starting and ending of columns in memory, in that order.

Insert/Delete Line Character

Generally, there are two kinds of programmable 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 by 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 versus 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. 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.
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 movement 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 the **smir/rmir** capabilities and the **ich1** capability can be given, and both will be used. The **ich** capability, with one parameter, ***n***, will repeat the effects of **ich1** ***n*** times.

Highlighting, Underlining, and Visual Bells

If your terminal has one or more kinds of display attributes such as highlighting, underlining, and visual 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. Visual bells include methods such as flashing the screen. The following capabilities describe highlighting, underlining, and visual 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 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, the **sgr0** turns off all attributes. Otherwise, if the terminal allows a sequence to set arbitrary combinations of modes, the **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. (The **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 visual bell capabilities. The **smcup** enters this mode, while the **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 when the **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 **kcub1**, **kcuf1**, **kcud1**, 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 addittion, 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**, **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 the **ht** (usually **Ctrl-I**). A "backtab" command which moves left toward the previous tab stop can be given as the **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 be use the **ht** or the **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 tabs 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 the **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 codes consistent with the rest of the **terminfo** file 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 analagously given as **rs1**, **rs2**, **rf**, and **rs3**, analagous 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 loggin 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 the **tbc** (clear all tab stops) and the **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 the **is2** or the **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 **cr**, **ind**, **cub1**, **ff**, and **tab** capabilities cause the appropriate delay bits to be set in the terminal driver. If the **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of the **pb**.

Miscellaneous Strings

If the terminal requires other than a null (zero) character as a pad, then this can be given as the **pad** string. 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 **hs** capability 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 the **tsl** and the **fsl**. (The **fsl** must leave the cursor position in the same place it was before the **tsl**. If necessary, the **sc** string and the **rc** string can be included in **tsl** and **fsl** to get this effect.) The **tsl** parameter 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 **eslok** flag 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 (possibley because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the **wsl** numeric parameter.

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 **rep** parameterized string. 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 the **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 the **smm** and the **rmm**.

If the terminal has more lines of memory than will filt on the screen at once, the number of lines of memory can be indicated with the **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: the **mc0** prints the contents of the screen, the **mc4** turns off the printer, and the **mc5** turns on the printer. When the printer is on, all text sent to the terminal is sent to 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 of the **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 the **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as the **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 the **pfkey** causes pressing the given key to be the same as the user typing the given string; the **pfloc** causes the string to be executed by the terminal in local mode; and the **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 the **hz**.

Terminals that ignore a line-feed character immediately after an **am** wrap should indicate the **xenl**.

If the **el** is required to get rid of standout (instead of merely writing normal text on top of it), the **xhp** should be given.

Terminals for which tabs turn all characters moved to blanks should indicate the **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 the **xsb**, indicating that the **F1** key is used for ESC and the **F2** key is used for Ctrl–C.

Other specific terminal problems can be corrected by adding more capabilities of the form **xx**.

Similar Terminals

If two terminals are very similar, one can be defined as being just like the other with certain exceptions. The **use** string capability can be given with the name of the similar terminal. The capabilities given before the use capability overrides those in the terminal type called by the use capability. 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** capability or the **rmkx** capability, and hence does not turn on the function key lables when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Data Base File Names

Compiled **terminfo** file descriptions are placed in subdirectories under the /usr/lib/terminfo file in order to avoid performing linear searches through a single directory containing all of the **terminfo** file description files. A given description file is stored in the **/usr/lib/terminfo/*c*/*name*** file, where *name* is the name of the terminal, and *c* is the first letter of the terminal name. For example, the compiled descripton 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 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;%;%?%p6t1;%;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,
    rmul1=\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, kimr=\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,
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/lib/terminfo/?/*	Compiled terminal capability data base.

Related Information

The **curses subroutine library**, **extended curses subroutine library**, **printf**, **fprintf**, **sprintf**, **NLprintf**, **NLfprintf**, or **NLsprintf** subroutines.

The **termdef** command, **tic** command.

user File

Purpose

Contains extended user attributes.

Description

The **/etc/security/user** file is an ASCII file that contains stanzas with extended user attributes. Each stanza is identified by a user name, followed by a : (colon), and contains comma-separated attributes in the *Attribute=Value* form. If an attribute is not defined for a user, either the default stanza or the default value for the attribute is used. For an example of a stanza, see the Examples section.

Each attribute is ended by a new line character, and each stanza is ended by an additional new line character.

Each stanza can have the following attributes:

admin Indicates whether the user is an administrator. Possible values are:

- **yes** The user is an administrator. Only the root user can change the attributes of this user. You can substitute the **true** keyword or the **always** keyword for the **yes** keyword.
- **no** The user is not an administrator. You can substitute the **false** keyword or the **never** keyword for the **no** keyword. This is the default value.

auth1 Defines the primary methods used to authenticate the user. The value is a list of comma-separated *Method*;*Name* pairs, with *Method* specifying the name of the authentication method, and *Name* specifying the user to be authenticated. The SYSTEM method, local password authentication, is the default value. The NONE method indicates that no primary authentication check is made. Any other *Method* must be the name of a stanza in the **/etc/security/login.cfg** file. If no principle *Name* is specified, the name of the invoker of the process is used.

auth2 Defines the secondary authentication methods for the user. These methods are invoked by the **login** command if primary authentication is successful. The value is a list of comma-separated *Method*;*Name* pairs, with *Method* specifying the name of the authentication method, and *Name* specifying the user to be authenticated. The NONE method indicates that no secondary authentication check is made; this is the default value. If the *Name* parameter is not specified, the name of the invoker of the process is used. The log in will not fail if a secondary authentication method is not successful.

daemon Defines whether the user can execute programs using the **cron** daemon or the **src** (system resource controller) daemon. Possible values are:

- **yes** The user can execute programs using the **cron** or **src** daemon. You can substitute the **true** keyword or the **always** keyword for the **yes** keyword.

no The user cannot execute programs using the **cron** or **src** daemon. You can substitute the **false** keyword or the **never** keyword for the **no** keyword.

expires Defines the expiration date for the user account, using the *MMDDhhmmyy* format, where *MM* = month, *DD* = day, *hh* = hour, *mm* = minute, and *yy* = last 2 digits of the year. All characters are numeric. See the **date** command for more information. If the value is 0, the account does not expire. The default value is 0.

login Defines whether local log ins are enabled for the user. Possible values are:

yes Local log ins are defined. You can substitute the **true** keyword or the **always** keyword for the **yes** keyword. This is the default value.

no Local log ins are not defined. You can substitute the **false** keyword or the **never** keyword for the **no** keyword.

rlogin Defines whether the user account can be accessed by remote log ins with the **rlogin** command. Possible values are:

yes The user account can be accessed remotely. You can substitute the **true** keyword or the **always** keyword for the **yes** keyword. This is the default value.

no The user cannot be accessed remotely. You can substitute the **false** keyword or the **never** keyword for the **no** keyword.

su Defines whether other users can switch to this user account with the **su** command. Possible values are:

yes Other users can switch to this account. You can substitute the **true** keyword or the **always** keyword for the **yes** keyword. This is the default value.

no Other users cannot switch to this account. You can substitute the **false** keyword or the **never** keyword for the **no** keyword.

sugroups Defines which groups can switch to this user account with the **su** command. The value is a comma separated list of valid group names on the system. If a group has no access, the name is prefixed by an ! (exclamation point). The keyword **ALL** means that all groups have access. The default value is **ALL**.

tpath Defines the user's trusted path characteristics. Possible values are:

nosak The secure attention key (SAK) is disabled for all processes run by this user. This value should be used for accounts that transfer binary data that may contain the SAK sequence.

notsh The user cannot invoke the trusted shell on the trusted path. If the secure attention key (SAK) sequence is given after a user logs in, the login session ends.

always The user can execute only trusted processes. This implies that the user's initial program is in the trusted shell or some other trusted process.

on The user has standard trusted path characteristics and can invoke a trusted shell in the trusted path with the secure access key (SAK) sequence. This is the default value.

ttys Defines which terminals can access the user account. The format is a list of comma-separated terminal-device path names. If a terminal cannot access the account, the name is prefixed by an ! (exclamation point). The keyword **ALL** means that access is allowed from all terminals. The default value is **ALL**.

umask The permissions a file has when the user creates it. The value is a three-digit octal number (*nnn*) that represents the read (r), write (w), and execute (x) permissions for the file owner, file group, and other users.

Access to this file should be through the commands and subroutines defined for this purpose.

The **mkuser** command creates an entry for each new user in the **/etc/security/user** file, initializing the attributes defined in the **/etc/security/mkuser.default** file. To change attribute values, use the **chuser** command. To display the attributes and their values, use the **lsuser** command. To remove an attribute, use the **rmuser** command.

To write programs that affect attributes in the **/etc/security/user** file, use the subroutines listed in Related Information.

Security

Access Control: This file should grant read (r) access only to the root user and members of the security group. Access for other users and groups depends upon the security policy for the system. Only the root user should have write (w) access.

Auditing Events:

Event	Information
S_USER_WRITE	filename

Example

A typical stanza looks like the following example for user `dhs`:

```
dhs:
        login = true
        rlogin = false
        ttys = /dev/console
        sugroups = security,!staff
        expires = 0531010090
        tpath = on
        admin = true
        auth1 = SYSTEM,METH2;dhs
```

Implementation Specifics

This command is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/security/user	Specifies the path to the file.
/etc/group	Contains the basic group attributes.
/etc/security/group	Contains the extended attributes of groups.
/etc/passwd	Contains the basic user attributes.
/etc/security/passwd	Contains password information.
/etc/security/environ	Contains the environment attributes of users.
/etc/security/limits	Contains the process resource limits of users.
/etc/security/audit/config	Contains audit system configuration information.

Related Information

The **chuser** command, **lsuser** command, **mkuser** command, **rmuser** command.

The **getuserattr** subroutine, **putuserattr** subroutine, **setuserdb** subroutine, **enduserdb** subroutine.

For more information about the identification and authentication of users, discretionary access control, the trusted computing base, and auditing, refer to Security Introduction in *General Concepts and Procedures*.

vfs File

Purpose

Describes the virtual file systems (VFS) installed on the system.

Description

The **vfs** file describes the virtual file systems (VFS) installed on the system. The name, type number, and file system helper program are among the types of information listed in the file. Commands, such as the **mount** command, the **fsck** command (file system check), and the **mkfs** command (make file system), use this information.

The **vfs** file is an ASCII file, with one record per line. The following examples are three types of lines in the **vfs** file:

- Comments

  #This is a comment.

  # Comments begin with a pound sign symbol (#).

  # Blank lines are ignored.

  #The following only locally defines the default **vfs** file.

- General control

  ```
  %defaultvfs jfs nfs
  ```

  The fields for the `%defaultvfs` control line are:

  %defaultvfs Identifies the control line.

  `jfs` Indicates the default local virtual file system.

  `nfs` Indicates the remote virtual file system (optional).

- Entries

```
#Name    Type    Mount Helper       Fs. helper

jfs       3      none                /etc/helpers/v3fshelper
nfs       2      /etc/nfsmnthelp     none
cdrfs     5      none                none
```

The comments, which are designated by a # (comment character), are in text for explanatory purposes. The general control lines, which are designated by a % (percent) character, configure the actions of the **mount** command, **umount** command, **mkfs** command, **fsck** command, **fsdb** command, **df** command, **ff** command. For example, a line like `%defaultvfs` indicates the default local virtual file system is used if no virtual file system is specified on the **mount** command or in the **/etc/filesystems** file. The entry is the name of the virtual file system as it is in the file. If a second entry is listed on the same line, it is taken to be the default remote virtual file system. The `%defaultvfs` control line may leave off the remote virtual file system specification.

The virtual file system entries take the following form:

name Canonical name of this type of virtual file system.

type Decimal representation of the virtual file system type number for the virtual file system.

mnt_helper Path name of the mount helper program of this virtual file system. If a mount helper is not required, the entry should be displayed as none. If this path name does not begin with a slash, it is relative to the **/etc/helpers** directory.

fs_helper Path name of the file system helper program of this virtual file system. If a file system helper is not required, the entry should be none. If this path name does not begin with a slash, it is relative to the **/etc/helpers** directory.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/filesystems	Lists the known file systems and defines their characteristics.
/etc/vfs	Contains descriptions of virtual file system types.

Related Information

The **filesystems** file format.

The **mount** command, **umount** command, **mkfs** command, **fsck** command,**df** command, **ff** command, **fsdb** command, **lsvfs** command, **chvfs** command, **rmvfs** command, **crvfs** command.

The File Systems Overview in *General Concepts and Procedures* explains file system types, management, structure, and maintenance.

vgrindefs File

Purpose

Contains the language definition database for the **vgrind** command.

Description

The **vgrindefs** file contains all the language definitions for the **vgrind** command.

Fields

The following table contains the name and description of each field:

Name	Type	Description
pb	str	Regular expression for the start of a procedure
bb	str	Regular expression for the start of a lexical block
be	str	Regular expression for the end of a lexical block
cb	str	Regular expression for the start of a comment
ce	str	Regular expression for the end of a comment
sb	str	Regular expression for the start of a string
se	str	Regular expression for the end of a string
lb	str	Regular expression for the start of a character constant
le	str	Regular expression for the end of a character constant
tl	bool	Present means procedures are only defined at the top lexical level
oc	bool	Present means upper and lowercase are equivalent
kw	str	A list of keywords separated by spaces

Regular Expressions

The **vgrindefs** file uses regular expressions which are very similar to those of the **ex** command and the **lex** command. The characters '^', '$', ':', and '\' are reserved characters and must be "quoted" with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

$	The end of a line
^	The beginning of a line
\d	A delimiter (space, tab, newline, start of line)
\a	Matches any string of symbols (like .* in the **lex** command)
\p	Matches any alphanumeric name. In a procedure definition (pb), the string that matches this symbol is used as the procedure name.
()	Grouping
|	Alternation
?	Last item is optional

\e Preceding any string, means that the string does not match an input string if the input string is preceded by an escape character (\). Typically used for languages (like C) which can include the string delimiter in a string by escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like "(tramp|steamer)flies?" would match "tramp", "steamer", "trampflies", or "steamerflies".

Keyword List

The keyword list is just a list of keywords in the language separated by spaces. If the "oc" boolean is specified, indicating that upper and lowercase are equivalent, then all the keywords should be specified in lowercase.

Example

The following entry, which describes the C language, is typical of a language entry:

```
C|c:    :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
        :lb=':le=\e':tl:\
        :kw=asm auto break case char continue default do
         double else enum\
        extern float for fortran goto if int long register
         return short\
        sizeof static struct switch typedef union unsigned
         while #define\
        #else #endif #if #ifdef #ifndef #include #undef # define
         else endif\
        if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it.) Thus the C language could be specified to the **vgrind** command as "c" or "C".

Entries can continue onto multiple lines by giving a \ (backslash) as the last character of a line. Capabilities in the **vgrindefs** file are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list.

Implementation Specifics

This file is part of Formatting Tools in the Text Formatting System of AIX for RISC System/6000.

File

/usr/lib/vgrindefs File containing terminal descriptions.

Related Information

The **ex** command, **lex** command, **vgrind** command, and **troff** command.

BNU audit File

Purpose

Contains debug messages from the **uucico** daemon.

Description

The **/usr/spool/uucp/.Admin/audit** file contains debug messages from the **uucico** daemon when it is invoked as a result of a call from another system. If the **uucico** daemon is invoked from the local system, the debug messages are sent to either the **/usr/spool/uucp/.Admin/errors** file or to standard output.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/spool/uucp/.Admin directory	Contains the **audit** file and other BNU administrative files.

Related Information

Working with BNU Log Files in *Communication Concepts and Procedures.*

The **cron** daemon, **uucico** daemon.

The **uudemon.cleanu** command.

BNU Command (C.*) Files

Purpose

Contain file transfer directions for the **uucico** daemon.

Description

Command (**C.***) files contain the directions that the Basic Networking Utilities (BNU) **uucico** daemon follows when transferring files. The full path name of a command file is a form of the following:

/usr/spool/uucp/*SystemName***/C.***SystemNameNxxxx*

The *SystemName* indicates the name of the remote system. The *N* character represents the grade of the work, and the *xxxx* notation is the four-digit hexadecimal transfer-sequence number: for example, `C.merlinC3119`.

The grade of the work specifies when the file is to be transmitted during a particular connection. The grade notation has the following characteristics:

- It is a single number (0–9) or letter (A–Z, a–z).
- Lower sequence characters cause the file to be transmitted earlier in the connection than do higher sequence characters. Sequence is established using ASCII order, beginning with 0 (zero) and ending with z.
- The number 0 is the highest grade (that is, the lowest character in the sequence), signifying the earliest transmittal; z is the lowest grade, specifying the latest transmittal.
- The default grade is N.

A command file consists of a single line that includes the following kinds of information in the following order:

1. An **S** (send) or **R** (receive) notation.

   Note: A send command file is created by the **uucp** or **uuto** commands; a receive command file is created by the **uux** command.
2. The full path name of the source file being transferred. A receive command file, does not include this entry.
3. The full path name of the destination file, or a path name preceded by *~user*, where *user* is a login name on the specified system. Here, the ~ (tilde) is shorthand for the name of the user's home directory.
4. The sender's login name.
5. A list of the options, if any, included with the **uucp**, **uuto**, or **uux** command.
6. The name of the data file associated with the command file in the spooling directory. This field must contain an entry. If one of the data-transfer commands (such as the **uucp** command with the default **–c** flag) does not create a data file the BNU program instead creates a placeholder with the name **D.0** for send files, or **dummy** for receive files.
7. The source file permissions code, specified as a three-digit octal number (for example, 777).
8. The login name of the user on the remote system who is to be notified when the transfer is complete.

Examples

Examples of Two Send Command Files

1. The send command file `/usr/spool/uucp/venus/C.heraN1133`, created with the **uucp** command, contains the following fields:

   ```
   S /u/amy/f1 /usr/spool/uucppublic/f2 amy –dC D.herale73655 777 lg
   h
   ```

   where:

   a. `S` denotes that the **uucp** command is sending the file.

   b. The full path name of the source file is `/u/amy/f1`.

   c. The full path name of the destination is `/usr/spool/uucppublic/f2`, where `/usr/spool/uucppublic` is the name of the BNU public spooling directory on the remote computer and `f2` is the new name of the file.

      Note: The destination name may be abbreviated as `~uucp/f2`. Here, the ~ (tilde) is a shorthand way of designating the public directory.

   d. The person sending the file is `amy`.

   e. The sender entered the **uucp** command with the **–C** flag, specifying that the **uucp** command program should transfer the file to the local spooling directory and create a data file for it. (The **–d** flag, which specifies that the command should create any intermediate directories needed to copy the source file to the destination, is a default.)

   f. The name of the data (**D.***) file is `D.herale73655`, which the **uucp** command assigns.

   g. The octal permissions code is `777`.

   h. The `lgh` login name of the user on system `hera`, who is to be notified of the file arrival.

2. The `/usr/spool/uucp/hera/C.zeusN3130` send command file, produced by the **uuto** command, is as follows:

   ```
   S /u/amy/out ~/receive/msg/zeus amy –dcn D.0 777 msg
   ```

   The `S` denotes that the `/u/amy/out` source file was sent to the `receive/msg` subdirectory in the public spooling directory on system `zeus` by user `amy`.

   Note: The **uuto** command creates the **receive/msg** directory if it does not already exist.

   The **uuto** command used the default flags **–d** (create directories), **–c** (transfer directly, no spooling directory or data file), and **–n** (notify recipient). The `D.0` notation is a placeholder, `777` is the permissions code, and `msg` is the recipient.

Example of a Receive Command File

3. The format of a receive command file is somewhat different from that of a send command file. When files required to run a specified command on a remote system are not present on that system, the **uux** command creates a receive command file.

   For example, the following command:

   ```
   uux – "diff /u/amy/out hera!/u/amy/out2 > ~uucp/DF"
   ```

   produces the `/usr/spool/uucp/zeus/C.heraR1e94` receive command file.

   Note: The command in this example invokes the **uux** command to run a **diff** command on the local system, comparing file `/u/amy/out` with file `/u/amy/out2`, which

is stored on remote system `hera`. The output of the comparison is placed in the `DF` file in the public directory on the local system.

The actual receive command file looks like this:

```
R /u/amy/out2 D.hera1e954fd amy - dummy 0666 amy
```

The `R` denotes a receive file. The **uucico** daemon, called by the **uux** command, gets the `/u/amy/out2` file from system `hera` and places it in a data file called `D.hera1e954fd` for the transfer. Once the files are transferred, the **uuxqt** daemon executes the command on the specified system.

User `amy` issued the **uux** command with the **-** (minus sign) flag, which makes the standard input to the **uux** command the standard input to the actual command string. No data file was created in the local spooling directory, so the BNU program uses `dummy` as a placeholder. The permissions code is `666` (the BNU program prefixes the three-digit octal code with a `0`), and user `amy` is to be notified when the command has finished executing.

Implementation Specifics

These files are part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp/Permissions file	Describes access permissions for remote systems.
/usr/lib/uucp/Systems file	Describes accessible remote systems.
/usr/spool/uucp/*SystemName* directory	Contains BNU command, data, and execute files.
/usr/spool/uucp/*SystemName***/D.*** files	Contain data to be transferred.
/usr/spool/uucppublic/* directories	Contain transferred files.

Related Information

The **cron** daemon, **uucico** daemon, **uusched** daemon, **uuxqt** daemon.

The **uucp** command, **uudemon.cleanu** command, **uupick** command, **uuto** command, **uux** command.

BNU Data (D.*) Files

Purpose

Contain data to be sent to remote systems.

Description

Data (**D.***) files contain the data to be sent to remote systems by the Basic Networking Utilities (BNU) **uucico** daemon. The full path name of a data file is a form of the following:

/usr/spool/uucp/*SystemName***/D.***SystemNamexxxx###*

where the *SystemName* directory and the *SystemName* portion of the file name indicate the name of the remote system. The *xxxx###* notation is the hexadecimal sequence number of the command (**C.***) file associated with that data file, for example: `D.venus471afd8`.

After a set period of time (specified by the **uusched** daemon), the **uucico** daemon transfers the data file to the designated system. It places the original data file in a subdirectory of the BNU spooling directory named **/usr/spool/uucp/***SystemName*, where the *SystemName* directory is named for the computer that is transmitting the file, and creates a temporary (**TM.***) file to hold the original data file.

After receiving the entire file, the BNU program takes one of the three following actions:

- If the file was sent with the **uucp** command and there were no transfer problems, the program immediately renames the **TM.*** file with the appropriate data file name, such as `D.venus471afd8`, and sends it to the specified destination.
- If the file was sent with the **uuto** command, the BNU program also renames the temporary data file with the appropriate **D.*** file name. The program then places the data file in the **/usr/spool/uucppublic** public directory, where the user receives the data file and handles it with one of the **uupick** command options.
- If there were transfer problems (such as a failed login or an unavailable device), the temporary data file remains in the spooling subdirectory. The **uudemon.cleanu** command, a shell procedure, removes these files automatically at specified intervals. They can also be removed manually.

Implementation Specifics

These files are part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp/Systems file	Describes accessible remote systems.
/usr/spool/uucp/*SystemName* directory	Contains BNU command, data, execute files.
/usr/spool/uucp/*SystemName***/C.*** files	Contain instructions for file transfers.
/usr/spool/uucp/*SystemName***/TM.*** files	Store data files temporarily after they have been transferred to a remote system.
/usr/spool/uucppublic/* directories	Contain files that BNU program has transferred.

Related Information

The **uucico** daemon, **uusched** daemon, **uuxqt** daemon.

The **uucp** command, **uudemon.cleanu** command, **uupick** command, **uuto** command, **uux** command.

BNU errors File

Purpose

Contains a record of **uucico** daemon errors.

Description

The **/usr/spool/uucp/.Admin/errors** file contains a record of **uucico** daemon errors that the Basic Networking Utilities (BNU) program cannot correct. For example, if the **uucico** daemon is unable to access a directory that is needed for a file transfer, the BNU program records this in the **errors** file.

If debugging is enabled for the **uucico** daemon, the BNU program sends the error messages to standard output instead of to the **errors** file.

Example

Following is the text of an error which might appear in the **errors** file:

```
ASSERT ERROR (uucico) pid: 303 (7/18-8:25:09) SYSTAT OPEN FAIL /usr/
spool/uucp/.Status/ (21) [SCCSID: @(#)systat.c   7.2 87/07/08 16:43:
37, FILE: systat.c, LINE:100]
```

This error occurred on July 18 at 8:25:09 a.m. [`(7/18-8:25:09)`] when the **uucico** daemon, running as process 303 [`(uucico) pid: 303`], could not open the **.Status** directory [`SYSTAT OPEN FAIL /usr/spool/uucp/.Status/`]. To prevent this error from occurring again, you should make sure the AIX permissions for the **.Status** directory are correct. It should be owned by the **uucp** login ID and group **uucp**, with permissions of 777 (read, write, and execute for owner, group, and all others).

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/spool/uucp/.Admin directory	Contains the **errors** file and other BNU administrative files.
/usr/spool/uucp/.Status/*SystemName* file	Lists the last time a remote system was contacted and the minimum time until the next retry.

Related Information

The **uucico** daemon.

The **uudemon.cleanu** command.

BNU Execute (X.*) Files

Purpose

Contain instructions for running commands that require the resources of a remote system.

Description

The Basic Networking Utilities execute (**X.***) files contain instructions for running commands that require the resources of a remote system. They are created by the **uux** command.

The full path name of a **uux** command execute file is a form of the following:

/usr/spool/uucp/*SystemName***/X.***RemoteSystemNxxxx*

where the *SystemName* directory is named for the local computer and the *RemoteSystem* directory is named for the remote system. The *N* character represents the grade of the work, and the *xxxx* notation is the four-digit hexadecimal transfer-sequence number; for example, `X.zeusN2121`.

Note: The grade of the work specifies when the file is to be transmitted during a particular connection. The grade notation is a single number (0–9) or letter (A–Z, a–z). Lower sequence characters cause the file to be transmitted earlier in the connection than do higher sequence characters. The number 0 is the highest grade, signifying the earliest transmittal; z is the lowest grade, specifying the latest transmittal. The default grade is N.

Standard Entries in an Execute File

An execute file consists of several lines, each with an identification character and one or more entries:

Line	Format and Description
User Line	**U** *UserName SystemName* Specifies the login name of the user issuing the **uux** command and the name of the system from which the command was issued.
Error Status Line	**N** or **Z** Indicates the error status. **N** Indicates that a failure message is *not* sent to the user issuing the **uux** command if the specified command does not execute successfully on the remote system. **Z** Indicates that a failure message is sent to the user issuing the **uux** command if the specified command does not execute successfully on the remote system.
Requester's Name	**R** *UserName* Specifies the login ID of the user requesting the remote command execution.
Required File Line	**F** *FileName* Contains the names of the files required to execute the specified command on the remote system. The *FileName* parameter can be either the complete path name of the file, including the unique

transmission name assigned by the BNU program, or simply the transmission name without any path information.

The Required File Line can contain zero or more file names. The **uuxqt** daemon checks for the existence of all listed files before running the specified command.

Standard Input Line — **I** *FileName*

Specifies the standard input to be used.

The standard input is either specified by a **<** (less than) symbol in the command string, or inherited from the standard input of the **uux** command if that command was issued with the **–** (minus sign) flag.

If standard input is specified, the input source is also listed in an **F** (Required File) line. If standard input is not specified, the BNU program uses the **/dev/null** device file.

Standard Output Line — **O** *FileName SystemName*

Specifies the names of the file and system that are to receive standard output from the command execution. Standard output is specified by a **>** (greater than) symbol within the command string. (The **>>** sequence is not valid in **uux** commands.) As is the case with standard input, if standard output is not specified, the BNU program uses the **/dev/null** device file.

Command Line — **C** *CommandString*

Gives the command string that the user requests to be run on the specified system. The BNU program checks the **/usr/lib/uucp/Permissions** file on the designated computer to see whether the login ID can run the command on that system.

All required files go to the execute file directory, usually **/usr/spool/uucp/.Xqtdir**. After execution, the standard output is sent to the requested location.

Examples

1. User `amy` on local system `zeus` issued the following command:

   ```
   uux – "diff /u/amy/out hera!/u/amy/out2 > ~uucp/DF"
   ```

   The command in this example invokes the **uux** command to run a **diff** command on the local system, comparing file `/u/amy/out` with file `/u/amy/out2`, which is stored on remote system `hera`. The output of the comparison is placed in the **DF** file in the public directory on the local system.

   The preceding command produces the `/usr/spool/uucp/hera/X.zeusN212F` execute file; which contains the following information:

   ```
   U amy zeus
   # return status on failure
   Z
   # return address for status or input return
   R amy
   F /usr/spool/uucp/hera/D.herale954fd out2
   O ~uucp/DF zeus
   C diff /u/amy/out out2
   ```

The user line identifies user amy on system zeus. The error-status line indicates that amy will receive a failure status message if the **diff** command fails to execute. The requestor is amy, and the file required to execute the command is the following data file:

```
/usr/spool/uucp/hera/D.herale954fd out2
```

The output of the command is to be written to the public directory on system zeus with the file name DF. (Remember that ~uucp is the shorthand way of specifying the public directory.) The final line is the command string that user amy entered with the **uux** command.

2. Following is another example of an execute file:

```
U uucp hera
# don't return status on failure
N
# return address for status or input return
R uucp
F D.hera5eb7f7b
I D.hera5eb7f7b
C rmail amy
```

This indicates that user uucp on system hera is sending mail to user amy, who is also working on system hera.

Implementation Specifics

These files are part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp/Permissions file	Describes access permissions for remote systems.
/usr/lib/uucp/Systems file	Describes accessible remote systems.
/usr/spool/uucp/*SystemName* directory	Contains BNU command, data, and execute files.
/usr/spool/uucp/*SystemName***/C.*** files	Contain instructions for transfers.
/usr/spool/uucp/.Xqtdir directory	Contains lists of commands that remote systems are permitted to execute.
/usr/spool/uucppublic/* directories	Contain files that have been transferred.

Related Information

The **uuxqt** daemon.

The **diff** command, **uux** command.

BNU Foreign File

Purpose

Logs contact attempts from unknown systems.

Description

The **/usr/lib/uucp/.Admin/Foreign** file lists access attempts by unknown systems. The **/usr/lib/uucp/remote.unknown** shell script appends an entry to the **Foreign** file each time a remote computer that is not listed in the local **/usr/lib/uucp/Systems** file attempts to communicate with that local system.

Someone with root user authority can customize entries in the **Foreign** file to fit the needs of a specific site by modifying the **remote.unknown** shell script.

Example

1. Following is a sample entry in the **Foreign** file:

   ```
   Wed Sep 20 20:38:22 CDT 1989: call from the system merlin
   ```

   System `merlin`, which is not listed in the **/usr/lib/uucp/Systems** file, attempted to log in on September 20 at 20:38 hours (10:38 p.m.). BNU did not allow the unknown system to log in.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp/Permissions file	Describes access permissions for remote systems.
/usr/lib/uucp/Systems file	Describes accessible remote systems.
/usr/lib/uucp/remote.unknown file	Records contacts from unknown systems in the **Foreign** file.
/usr/spool/uucp/.Admin directory	Contains the **Foreign** file and other BNU administrative files.

Related Information

The **cron** daemon, **uucico** daemon, **uuxqt** daemon.

The **uucp** command, **uudemon.cleanu** command, **uux** command.

BNU remote.unknown File

Purpose

Logs access attempts by unknown remote systems.

Description

The **/usr/lib/uucp/remote.unknown** file is a shell script. It is executed by the Basic Networking Utilities (BNU) program when a remote computer that is not listed in the local **/usr/lib/uucp/Permissions** file attempts to communicate with that local system. The BNU program does not permit the unknown remote system to connect with the local system. Instead, the **remote.unknown** shell procedure appends an entry to the **/usr/spool/uucp/.Admin/Foreign** file.

Modify the **remote.unknown** file to fit the needs of your site. For example, to allow unknown systems to contact your system, remove the execute permissions for the **remote.unknown** file. You can also modify the shell script to send mail to the BNU administrator or to recognize certain systems and reject others.

Note: Only someone with root user authority can edit the **remote.unknown** file, which is owned by the **uucp** program login ID.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **remote.unknown** file.
/usr/lib/uucp/Permissions file	Describes access permissions for remote systems.
/usr/spool/uucp/.Admin/Foreign file	Lists access attempts by unknown systems.

Related Information

How to Configure BNU and Understanding BNU Security in *Communication Concepts and Procedures*.

BNU Temporary (TM.*) Files

Purpose

Store data files during transfers to remote systems.

Description

The Basic Networking Utilities Temporary (**TM.***) files store data files during transfers to remote systems.

After a data (**D.***) file is transferred to a remote system by the **uucico** daemon, the BNU program places the file in a subdirectory of the BNU spooling directory named **/usr/spool/uucp/***SystemName*. Here the *SystemName* directory is named for the computer that is transmitting the file. The BNU program creates a temporary data file to hold the original data file.

The full path name of the temporary data file is a form of the following:

/usr/spool/uucp/*SystemName*/**TM.***xxPID*.**000**

where the *SystemName* directory is named for the computer that is sending the file, and **TM.***xxPID*.**000** is the name of the file; for example, `TM.00451.000`. The *PID* variable is the process ID of the job.

The **uucico** daemon normally deletes all temporary files when they are no longer needed. However, temporary files can also be removed using the **uucleanup** command with the **–T** flag.

Implementation Specifics

These files are part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp/Systems file	Describes accessible remote systems.
/usr/spool/uucp/*SystemName* directory	Contains BNU command, data, and execute files.
/usr/spool/uucppublic/* directories	Contain files that BNU has transferred.
/usr/spool/uucp/*SystemName*/**D.*** files	Contain data to be transferred.

Related Information

The **uucico** daemon.

The **uucp** command, **uucleanup** command, **uudemon.cleanu** command, **uupick** command, **uuto** command, **uux** command.

BNU xferstats File

Purpose

Contains information about the status of file transfer requests.

Description

The **/usr/spool/uucp/.Admin/xferstats** file contains information about the status of each Basic Networking Utilities (BNU) file transfer request. The **xferstats** file contains the following information:

- System name
- Name of the user requesting the transfer
- Date and time of the transfer
- Name of the device used in the transfer
- Size of the transferred file
- Length of time the transfer took.

Example

1. Following is a typical entry in the **xferstats** file:

   ```
   zeus!jim M (10/11-16:10:33) (C,9234,1) [-] -> 1167 / 0.100 secs
   ```

   A file was transferred by user `jim` to system `zeus` at 4:10 p.m. on the 11th of October. The file size was `1167` bytes and the transfer took `0.100` seconds to complete.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

File

/usr/spool/uucp/.Admin directory	Contains the **xferstats** file and other BNU administrative files.

Related Information

The **cron** daemon, **uucico** daemon, **uuxqt** daemon.

The **uucp** command, **uudemon.cleanu** command, **uux** command.

HCON e789_ctbl File

Purpose

Contains the default binary color definition table for HCON.

Description

The **/usr/lib/hcon/e789_ctbl** file contains the default color definition table for the AIX 3270 Host Connection Program/6000 (HCON) in binary form. The **/usr/lib/hcon/e789_ctbl.p** file contains the source mappings that were used to create the binary **/usr/lib/hcon/e789_ctbl** file.

Instances of the **e789_ctbl** file can also occur in users' **$HOME** directories. The color definition table can be customized using the **e789cdef** command. If the user issuing the **e789cdef** command does not specify a name for the new table, the command names the table **e789_ctbl** and places it in the user's **$HOME** directory. To use a customized table, an HCON user must specify the file name of the table in an HCON session profile.

Implementation Specifics

This file is part of the AIX 3270 Host Connection Program/6000 (HCON).

Files

/usr/lib/hcon directory	Contains HCON files, including the **e789_ctbl** file.
/usr/lib/hcon/e789_ctbl.p file	Contains the source for the **/usr/lib/hcon/e789_ctbl** file.
/usr/lib/hcon/nls_names file	Contains the HCON highlighting, attribute, and color name definitions.

Related Information

Use the **e789cdef** command to customize your color definition file.

How to Customize the HCON Color Definition Table in *Communication Concepts and Procedures* discusses creating customized color tables.

To start SMIT for use with HCON, use the **smit hcon** command.

To set up a session profile for HCON, use SMIT or the **mkhcons** command. To modify an existing profile, use SMIT or the **chhcons** command.

HCON e789_ktbl File

Purpose

Contains the default binary keyboard definition table used by HCON.

Description

The **/usr/lib/hcon/e789_ktbl** file contains the default keyboard definition table used by the AIX 3270 Host Connection Program/6000 (HCON) in binary form. The **/usr/lib/hcon/e789_ktbl.p** file contains the source mappings that were used to create the binary **e789_ktbl** file.

HCON key names are mapped to specific keys on each supported keyboard. The HCON emulator program uses these key mappings to generate the correct key function on all the supported keyboards. HCON key mappings can be customized using the **e789kdef** command.

Instances of the **e789_ktbl** file can also occur in users' **$HOME** directories. The keyboard definition table can be customized using the **e789kdef** command. If the user issuing the **e789kdef** command does not specify a name for the new table, the command names the table **e789_ktbl** and places it in the user's **$HOME** directory. To use a customized table, an HCON user must specify the file name of the table in an HCON session profile.

Implementation Specifics

This file is part of the AIX 3270 Host Connection Program/6000 (HCON).

Files

/usr/lib/hcon directory	Contains HCON files, including the **e789_ktbl** file.
/usr/lib/hcon/e789_ktbl.p file	Contains the source for the default binary keyboard definition table.
/usr/lib/hcon/func_names file	Contains HCON function names.
/usr/lib/hcon/keynames file	Contains HCON keynames.

Related Information

Use the **e789kdef** command to customize your keyboard definition file.

How to Customize the HCON Keyboard Definition Table in *Communication Concepts and Procedures* discusses creating customized keyboard tables.

To start SMIT for use with HCON, use the **smit hcon** command.

To set up a session profile for HCON, use SMIT or the **mkhcons** command. To modify an existing profile, use SMIT or the **chhcons** command.

Mail aliases File

Purpose

Contains alias definitions for the **sendmail** command.

Description

This file contains the required aliases for the **sendmail** command. Do not change these defaults, as they are needed by the system. It is formatted as a series of lines in the form:

```
name: name_1, name_2, name_3,...
```

The `name` is the name to alias, and the `name_n` are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with # (pound signs) are comments.

Aliasing occurs only on local names.

System-wide aliases are always used to redirect mail. For example, if you receive mail at three different systems, you can use the **/usr/lib/aliases** file and redirect your mail to one of the systems.

Aliases can be defined to send mail to a distribution list. For example, you can send mail to all of the members of a project by sending mail to just a single name.

Normally the sender of a message is not included when the **sendmail** command expands an alias address. For example, if `amy` sends a message to alias `D998` and she is defined as a member of that alias, the **sendmail** command does not send a copy of the message to `amy`.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files **/usr/lib/aliasesDB/DB.dir** and **/usr/lib/aliasesDB/DB.pag** using the **newaliases** command. For the change to take effect, the **newaliases** command must be executed each time the aliases file is changed.

Implementation Specifics

This **/usr/lib/aliases** file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/lib/aliases	Contains system-wide aliases.
/usr/lib/aliasesDB	Contains the binary files created by the **newaliases** command.

Related Information

The **newaliases** command, **sendmail** command.

Understanding Mail Aliases, How to Build the Alias Database, and How to Create a Local System Alias in *Communication Concepts and Procedures*.

Mail sendmail.cf File

Purpose

Contains **sendmail** configuration file data.

Description

The configuration file contains the configuration information for the **sendmail** command. The configuration information includes such items as the host name and domain and the **sendmail** rule sets.

The configuration file has three major purposes:

- Initialize the environment for the **sendmail** command by setting the options.
- Define how the sendmail command rewrites addresses in messages.
- Define the set of instructions needed to deliver a message.

The configuration file entries consist of control lines, each of which begins with a single character command. Entries can continue onto multiple lines by placing a white space at the beginning of each subsequent line. Comment lines begin with the # (pound) character.

Commands and Operands

C*XWord1 Word2...*

Defines the class (*X*) of words that can be used to match on the left hand side of rewriting rules. Class specifiers may be any of the uppercase letters from the ASCII character set. Lowercase letters and special characters are reserved for system use.

D*XValue* Defines the macro (*X*) and its associated *Value*. Macros specifiers may be any of the uppercase letters from the ASCII character set. Lowercase letters and special characters are reserved for system use.

F*XFileName* [*Format*]

Reads the elements of the class (*X*) from *FileName* using an optional **scanf** format specifier. The format specifier can contain only one conversion specification. Only one class number is read for each line in *FileName*.

H[*?MFlags?*]*HeaderName: HeaderTemplate*

Defines the header format the **sendmail** command inserts into a message. Continuation lines are a part of the definition. The *HeaderTemplate* is macro-expanded before insertion into the message. If the *MFlags* are specified, and at least one of the specified flags is included in the mailer definition, then this header is automatically written to the output message. If the header appears in the input message, it is written to the output message regardless of the **MFlags** field.

M*Name,* [*Field=Value*]

Defines a mailer program where *Name* is the name of the mailer program and *Field*=Value defines the attributes of the mailer.

O*x*[*Value*] Sets option to *x*. If the option is a valued option, you must also specify *Value*. Options may also be selected from the command line.

P*Name=Number*

Defines values for the `Precedence:` header field. When *Name* is found in a message's `Precedence:` field, the message's precedence is set to *Number*. Higher numbers indicate higher precedences. Negative numbers indicate that error messages are not returned. The default *Number* is 0.

R*LeftHandSide RightHandSide Comments*

Defines a rewriting rule. One or more tab characters separate the three fields of this command. If space characters are used as field separators, the configuration option **J** must be set. The fields may contain embedded spaces, unless the **J** option is set. If the **J** option is set, the embedded spaces must be represented by the character defined in **J**. After the fields are separated, the character representing the space is changed to an actual space.

S*x* Sets the rule set currently defined to number *x*. If a rule set definition is begun more than once, the new definition overwrites the old.

T*User1 User2 ...* Defines the system-administrative user IDs. These IDs have permission to override the sender address using the **–f** flag. There can be more than one ID specified per line.

Special Macros

Macros are interpolated using **$***x*, where *x* is the name of the macro to be interpolated. Lowercase letters are reserved by the **sendmail** command.

Required Macros

Warning: Altering required macros may render the Mail Program unusable.

If you create a new **/usr/lib/sendmail** configuration file, be sure to include definitions for these macros which the **sendmail** daemon requires:

e macro Denotes the Simple Mail Transfer Protocol (SMTP) entry message.

j macro Denotes the official domain name for this site.

l macro Denotes the format of the From line (not the `From:` line).

n macro Denotes the name of the daemon (for error messages).

o macro Denotes the set of operators in addresses.

q macro Denotes the default format of sender address.

Other Macros

The **sendmail** command defines some macros for interpolation into argument variables for mailer programs or for other contexts. These macros are:

a macro The origination date in ARPANET form. The **$a** macro contains the time extracted from the `Date:` line of the message. If the incoming message has no `Date:` line, the **$a** macro contains the current time.

b macro The current date in ARPANET form. The **$b** macro equals the current date and time. This macro is used for postmarks.

c macro — The hop count. The hop count is the number of times the message has been processed.

d macro — The date in AIX (**ctime**) format.

f macro — The sender address. The **$f** macro is the sender address as seen from the host.

g macro — The sender address relative to the receiver. When mailing to a specific host, the **$g** macro contains the address of the sender relative to the receiver. For example, if the user, `newton`, at system, `appletree`, sends a message to `chopin@piano`, the **$f** macro equals `newton` and the **$g** macro equals `newton@appletree`.

h macro — The receiving host.

i macro — The queue ID of the host. The **$i** is useful for tracking messages if put into the message ID line.

p macro — The process ID of the **sendmail** command. The **$p** macro and the **$t** macro are used to create unique strings for the **Message_ID** field.

s macro — The host name of the sender.

t macro — A numeric representation of the current time. The **$p** macro and the **$t** macro are used to create unique strings for the **Message_ID** field.

u macro — The receiving user.

v macro — The version number of AIX. The **$v** macro is printed in the `Received:` header message and is useful for debugging.

w macro — The host name of the local site.

x macro — The full name of the sender. The name is determined by one of the the following:

- Full name passed as a flag to the **sendmail** command
- Value found in the **Full_Name** line of the header
- Value found in the comment field of a `From:` line
- Full name found in the **/etc/passwd** file if the message originates locally.

y macro — The terminal ID of the sender.

z macro — The home directory of the receiver.

MName, [Field=Value] Attributes for Sendmail.cf

Name is the name of the mailer and *Field=Value* defines the attributes of the mailer program. Allowable *Fields* and *Values* are:

A=*Argument* — Specifies information to be passed to the mailer. This field can be any string of words with embedded spaces allowed. Any or all of the words can be symbols. If you do not include this field, or the field does not contain the **$u** symbol, the **sendmail** command uses the SMTP to send messages to the mailer.

E=*EndOfLine* — Defines the end of line. The default is **newline**.

F=*Flags* Defines the meaning for the flags that the **sendmail** command recognizes. The valid flags are:

C Appends the sender domain. The sender domain ID is defined as everything from the first @ (at) character to the end of the sender address. This string is appended to header addresses which contain no @ symbols whenever mail is received from this mailer. This also applies to calculation of the **$g** macro and everything dependent on it.

D Requires a **Date** header line.

e This mailer is expensive to connect to. Avoid normal connection and, when necessary, connect during a queue run.

E Changes `From:` lines to **>From** in message bodies.

f Calls the mailer program with an **–f** flag followed by the expression of **$g**, if the following requirements are met:

- Specified mailer program is legitimate
- User ID is a system administrative or root ID
- Group ID is system.

F Requires a `From:` header line.

h Preserves uppercase in host names for this mailer program.

I Uses SMTP to communicate with another **sendmail** command and can use special protocol features.

l Performs final delivery on the local system.

L Limits the line length of a text line to less than 1000 characters. Any leading dot duplicated due to the **X** flag is not included in the count. Only allows 7-bit data to pass either way through the mailer program.

m Sends to multiple users on the same host in one transaction. when a **$u** macro occurs in the **A** part of the mailer program definition, that field will be repeated as necessary for all qualifying users.

M Requires a **Message-ID** header line.

n AIX-style `From:` lines on the front of the message are not inserted.

N Uses International Character Support. Only has meaning when used with the **L** flag. Allows 8-bit data to pass.

p Uses the *return-path* in the SMTP `MAIL FROM:` command rather than just the return address.

r Same as option **f** except a **–r** flag is generated.

s Strips quotation marks off of the address before calling the mailer.

S Does not reset user ID before calling the mailer.

u Preserves uppercase in users names for this mailer.

U Requires `From:` lines with UUCP-style **remote from** *host* on the end.

x Requires a **Full-Name** header line.

X Uses the hidden-dot algorithm. (Any line starting with a dot has an extra dot added at the beginning.) This ensures that the lines in the message containing a leading dot will not terminate the message prematurely.

M=*Max* Defines the maximum size of messages in bytes that the mailer handles.

P=*Path* Defines the full path name of the mailer on the local system. If the mailer uses the Simple Mail Transfer Protocol (SMTP), use the `[IPC]` string as the path.

R=*Recipient* Defines the rewrite rule set to be used on recipient addresses for this mailer.

S=*Sender* Defines the rewrite rule set to be used on sender addresses for this mailer.

Ox[Value] Attributes for Sendmail.cf

If the option is a valued option, you must also specify *Value*. The options and the possible values are described as follows:

A*File* Uses the named *File* as the alias file.

B*c* Sets the blank substitution character to the character specified in the parameter *c*. The **sendmail** command replaces spaces which are not quoted in addresses with this character. The supplied configuration file uses the . (period) for this character.

c Causes the **sendmail** command to queue messages for that mailer program without sending them if an outgoing mailer program is marked as expensive to use. The queue can run later when costs are lowest or when the queue is large enough to send the message efficiently.

d*x* Sets the delivery mode to *x*. Valid modes are:

- **b** Delivers in the background (asynchronously). This is the default setting.
- **i** Delivers interactively (synchronously).
- **q** Queues only the message and delivers it during queue run.

e*x* Sets error processing to mode *x*. Valid modes are:

- **e** Mails the error message to the user's mailbox, but always exits with a zero exit status (normal return).
- **m** Mails the error message to the user's mailbox.
- **p** Displays the error message on the terminal (default).
- **q** Discards the error message and returns the exit status only.
- **w** Writes the error message to the terminal if delivering in interactive mode; otherwise it mails the error message to the user's mail box.

f Saves `From` lines at the front of the messages. These lines are normally discarded. Causes all other headers to be regarded as part of the message body.

g*N* Sets the default group ID to use when calling mailers to the value specified by *Number*.

H*file* Specifies the name of the SMTP help file.

i Does not interpret a . (period) on a line by itself as a message terminator. Removes the excess period inserted at the beginning of a line by a remote mailer program, if mail is received through SMTP. In addition, if receiving mail through SMTP, any period at the front of a line followed by another period is removed. This is opposite of the action performed by the **X** mailer flag.

J*x* Allows spaces as well as tabs to separate the Left Hand Side (LSH) and Right Hand Side (RHS) of rewrite rules. In both the LHS and RHS, *x* must be used in place of embedded spaces. The default for *x* is _ (underscore). All instances of *x* are changed to spaces after the LHS and RHS are separated by the **sendmail** command. This option allows rewrite rules to be modified using an editor that replaces tabs with spaces.

L*number* Specifies the log level to be the value supplied in the *n* parameter. Each number in the following list includes the activities of all numbers of lesser value and adds the activity that it represents. Valid levels and the activities they represent are:

0	No logging
1	Major problems only
2	Message collections and failed deliveries
3	Successful deliveries
4	Messages being deferred
5	Placing messages in the queue
6	Unusual but benign incidents
9	Log internal queue ID to external message ID mappings
12	Several messages of interest when debugging
22	Verbose information regarding the queue.

m Includes the sender when sending a mail message to an alias that also lists the sender.

M*x Value* Defines macro *x* to have *Value*. This option is normally used only from the **sendmail** command line.

n Validates the Right Hand Side of aliases when performing the **newaliases** command.

N*NetworkName* Sets the name of the host network. The **sendmail** command compares the argument of an SMTP **HELO** command to *HostName.NetName* (value of *HostName* comes from the kernel). If these values do not match, it adds the *HostName.NetName* string to the `Received:` line in the message so that messages can be traced accurately.

o Indicates that this message can have headers with spaces between addresses. Without this option, the message has commas between addresses. If this option is set, an adaptive algorithm determines the header format in most cases.

P*address* Identifies the person who is to receive a copy of all returned mail.

Q*dir* Sets the directory in which to queue messages. The directory will be created if it does not exist.

r*time* Sets the time out for reads from a mailer program to the value specified by *time*. If no time out value is set, the **sendmail** command waits indefinitely for a mailer program to respond.

s Enqueues before delivery, even when in immediate delivery mode.

S*File*	Sets the mail statistics file to the *File*. Statistics are only collected if the file exists. This file must be created by the user.
T*time*	Sets the time out on messages in the queue to the specified *time*. After this interval, the **sendmail** command returns the message to the sender. The default is three days.
u*N*	Sets the default user ID to use when calling mailers to the value specified by *N*.
v	Runs in verbose mode.
Y	Delivers each message in the mail queue from a separate process. This option is not required and can, if used, increase overhead in the AIX environment.

Implementation Specifics

This **sendmail.cf** file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/lib/sendmail.cf	The configuration file for the **sendmail** command.
/usr/lib/sendmail.cfDB	The compiled version of the **sendmail** configuration file.

Related Information

The **edconfig** command, **newaliases** command, **sendmail** command.

The **/etc/passwd** file.

Understanding Sendmail and Understanding the sendmail.cf File in *Communication Concepts and Procedures*.

MH .maildelivery File

Purpose

Specifies actions to be taken when mail is received.

Description

The **$HOME/.maildelivery** file contains a list of actions that the **slocal** command performs on received mail. The **slocal** command reads the **$HOME/.maildelivery** file and performs the actions specified. To activate the **slocal** command:

1. Create a file called **.forward** in your home directory.
2. Place the following line in the **$HOME/.forward** file:

```
| /usr/lib/mh/slocal
```

Specify your own mail delivery instructions in the **$HOME/.maildelivery** file. Each line in the **$HOME/.maildelivery** file describes an action and the conditions under which the action should be performed. All of the five following parameters must be present in each line of the file. These parameters are separated by either commas or space characters:

Field Pattern Action Result "String"

Blank lines in the **.maildelivery** file are ignored. Put a # (pound) sign in the first column to indicate a comment. The file is always read completely, so several matches can be made with several actions. The **.maildelivery** file should be owned by the user, and the owner can be the only one with write access.

If the **.maildelivery** file cannot be found or does not deliver the message, the **/usr/lib/mh/maildelivery** file is used in the same manner. If the message has still not been delivered, it is delivered to the user's mail drop. The default mail drop is the **/usr/mail/$USER** file.

The MH package contains four standard programs that can be run as receive-mail hooks: the **rcvdist** command, **rcvpack** command, **rcvstore** command, and **rcvtty** command.

Parameters

The following list describes each parameter:

Field Specifies a header component to be searched to find a pattern to match the *Pattern* parameter. Specify one of the following values for the *Field* parameter:

Component	Specify the header component you want to be searched; for example, `From` or `cc`.
`*`	Matches everything.
`addr`	Searches whatever field was used to deliver the message to you.
`default`	Matches only if the message has not been delivered yet.
Source	Specifies the out-of-band sender information.

Pattern Specifies the character string to search for in the header component given by the *Field* parameter. For example, if you specified `From` in the *Field* parameter, the *Pattern* parameter might contain an address like `sarah@mephisto`.

The *Pattern* parameter is not case-sensitive. Thus, the character string matches any combination of uppercase and lowercase characters. Specify a dummy pattern if you use an * (asterisk) or specify `default` in the *Field* parameter.

Action Specifies an action to take with the message if it contains the pattern specified in the *Pattern* parameter. Specify the following values:

file or **>** Appends the message to the file specified with the "*String*" parameter. If the message can be written to the file, the action is considered successful. The `Delivery-Date:` header component is added to the message to indicate when the message was appended to the file.

pipe or **|** Pipes the message as standard input to the command specified with the "*String*" parameter. The shell interprets the string. If the exit status from the command is 0 (zero), the action is considered successful. Prior to being given to the shell, the string is expanded with the following built-in variables:

$(*Sender***)**	The return address for the message.
$(*Address***)**	The address that was used to deliver the message.
$(*Size***)**	The size of the message in bytes.
$(*ReplyTo***)**	Either the `Reply-To:` or `From:` header component of the message.
$(*Information***)**	Miscellaneous out-of-band information.

When a process is invoked with the pipe mechanism, the environment of the process is set as follows:

- User and group IDs are set to the recipient's IDs.
- Working directory is the recipient's directory.
- The value of the **umask** variable is 0077.
- Process has no **/dev/tty** special file.
- Standard input is set to the message.
- Standard output and diagnostic output are set to the **/dev/NULL** special file. All other file descriptors are closed. The **$USER**, **$HOME**, and **$SHELL** environmental variables are set appropriately; no other environment variables exist.

The amount of time the process is given to execute is:

(<bytes in message> x 60) + 300 seconds).

After that time, the process is terminated.

If the exit status of the program is 0 (zero), it is assumed that the action succeeded. Otherwise, failure is assumed.

qpipe or **^** — Similar to **pipe**, but executes the command directly after built-in variable expansion without assistance from the shell. If the exit status from the command is 0 (zero), the action is successful.

destroy — Destroys the message. This action always succeeds.

Result — Indicates how the action should be performed. You can specify one of the following values for this parameter:

A — Performs the action. If the action succeeds, the message is considered delivered.

R — Performs the action. Even if the action succeeds, the message is not considered delivered.

? — Performs the action only if the message has not been delivered. If the action succeeds, the message is considered delivered.

"*String*" — If you use the **file** value for the *Action* parameter, the "*String*" parameter specifies the file to which the message can be appended.

If you use the **pipe** or the **qpipe** value, the "*String*" parameter specifies the command to execute.

If you use the **destroy** value as the *Action* parameter, the "*String*" parameter is not used, but you must still include a dummy "*String*" parameter.

Note: To be notified that you have mail, you must specify the **rcvtty** command in the **.maildelivery** file.

Examples

The following are example lines in the **$HOME/.maildelivery** file:

1. To save a message in a particular file, use a line similar to the following line:

   ```
   From george file A george.mail
   ```

   This line directs the **slocal** command to search the `From` header line in messages. When the **slocal** command finds a message from `george`, it files the message in a file called `george.mail`.

2. To save a copy of a message in a file, use a line similar to the following:

   ```
   addr manager > R proj_X/statlog
   ```

This line directs the **slocal** command to search the address fields in messages. When it finds a message to the project manager, the **slocal** command files a copy of the message in a file called `proj_X/statlog`. The original message is not considered delivered (the `R` value), so the message is still treated as mail and you will be notified as usual.

3. To be notified that you have received mail, enter a line similar to the following:

   ```
   * – | R "/usr/lib/mh/rcvtty /u/sarah/allmail"
   ```

   In this example, the `/u/sarah/allmail` file contains the line:

   ```
   echo "You have mail\n"
   ```

   The `/u/sarah/allmail` file must have execute permission. When you have mail, the words `You have mail` will be displayed on your console.

4. To forward a copy of a message, enter a line similar to the following:

   ```
   addr manager | A "/usr/lib/mh/rcvdist amy"
   ```

   This line directs the **slocal** command to search the address fields in messages. When it finds a message to the project manager, the **slocal** command sends a copy of the message to `amy`. The original message is not affected. The action is always performed (the `A` value). The command that the **slocal** command reads to distribute the copy to another user is the **rcvdist** command.

5. To save any undelivered messages, enter a line similar to the following:

   ```
   default – > ? mailbox
   ```

   This line directs the **slocal** command to find all undelivered messages. The – (dash) is a placeholder for the *Pattern* parameter. The > (file symbol) instructs the **slocal** command to file the messages it finds. The `?` (question mark) instructs the **slocal** command to respond only to undelivered messages. The name of the file to store undelivered messages is `mailbox`.

Implementation Specifics

This file is part of Message Handler in BOS Extensions 1.

Files

$HOME/.forward	Searched by the **sendmail** command when mail is received.This file can contain either a path of a machine to which to forward mail or a line to start the **slocal** command.
/usr/mail/$USER	Provides the default mail drop.
/usr/lib/mh/slocal	Contains the **slocal** command that reads the **.maildelivery** file.
/usr/lib/mh/maildelivery	Contains the mail delivery instructions that the **slocal** command reads if none are specified in the **$HOME/.maildelivery** file.
$HOME/.maildelivery	Specifies mail-related actions for the **slocal** command to perform.

Related Information

The **rcvdist** command, **rcvpack** command, **rcvstore** command, **rcvtty** command, **sendmail** command, **slocal** command.

The **mtstailor** file.

MH .mh_profile File

Purpose

Customizes the Message Handler (MH) package.

Description

Each user of the MH package is expected to have a **$HOME/.mh_profile** file in the home directory. This file contains a set of user parameters used by some or all of the MH programs. Each line of the file has the following format:

Profile-Entry: Value

Profile Entries

Some entries have default values if they are not specified. The only required entry is `Path:`.

`Path:` Specifies the directory path for the user's MH directory, *UserMHDirectory*. The usual location is the **$HOME/Mail** directory. This information is stored in the MH profile. No default value is set.

`context:` Declares the location of the MH context file. This information is stored in the MH profile. The default value is the *UserMHDirectory*/**context** file.

`Current-Folder:` Keeps track of the current open folder. This information is stored in the context file. The default value for this entry is **inbox**.

`Previous-Sequence:` Names the sequences that should be defined as the *Messages* or *Message* parameter given to the program. If not present or empty, no sequences are defined. Otherwise, for each name given, the sequence is first set to 0 (zero), and then each message is added to the sequence. This information is stored in the MH profile. No default value is set.

`Sequence-Negation:` Defines the string that negates a sequence when prefixed to the name of that sequence. For example, if the `Sequence-Negation:` entry is set to `not`, then `not seen` refers to all the messages that are not a member of the sequence `seen`. This information is stored in the MH profile. No default value is set.

`Unseen-Sequence:` Names the sequences that are defined as those messages recently incorporated by the **inc** command. The **show** command removes messages from this sequence after they have been seen. If not present, or empty, no sequences are defined. Otherwise, for each name given, the sequence is first set to 0 (zero), and then each message is added to the sequence. This information is stored in the MH profile. No default value is set

`.mh_sequences:` Names in each folder the file that defines public sequences. To disable the use of public sequences, leave the value of this entry blank. This information is stored in the MH profile. The default value is **.mh_sequences**.

`atr-`*SequenceFolder*`:` Keeps track of the private sequence named *Sequence* in the specified *Folder*. (This information is stored in the context file. No default value is set.)

`Editor:` Defines the editor to be used by the **comp**, **dist**, **forw**, and **repl** commands. This information is stored in the MH profile. The default value is `prompter`.

`Msg-Protect:` Defines octal protection bits for message files. This information is stored in the MH profile. The default value is `0644`.

The **chmod** command explains the default values.

`Folder-Protect:` Defines protection bits for folder directories. This information is stored in the MH profile. The default value is `0711`.

The **chmod** command explains the default values.

Program`:` Sets default flags to be used whenever the MH program specified by the MH program field is invoked. For example, you can override the `Editor:` profile component, when replying to messages, by entering the following profile entry:

```
repl: -editor /bin/ed
```

This information is stored in the MH profile. No default value is set.

LastEditor`-next:` Specifies the editor that is the default editor after the editor specified by the `Editor:` field has been used. This takes effect at the `What now?` level of the **comp**, **dist**, **forw**, and **repl** commands. After the draft has been edited with the editor specified by the `Editor:` field, the default editor is set to be the editor specified by the *LastEditor*`-next:` field.

If you enter the **edit** command without a parameter to the `What now?` prompt, the editor specified by the *LastEditor*`-next:` field is used. This information is stored in the MH profile. No default value is set.

`Folder-Stack:` Defines the contents of the folder stack of the **folder** command. This information is stored in the context file. No default value is set.

`Alternate-Mailboxes:` Indicates to the **repl** and **scan** commands which addresses are really yours. In this way, the **repl** command knows which addresses should be included in the reply, and the **scan** command knows if the message really originated from you. Addresses must be separated by a comma.

The host names listed should be the official host names for the mailboxes you indicate. Local nicknames for hosts are not replaced with their official site names. If a host is not given for a particular address, that address on any host is considered to be your current address.

In addition, an * (asterisk) can be displayed at either end or both ends of the host mailbox to indicate pattern matching.

This information is stored in the MH profile. The default value is **$LOGNAME**.

`Draft-Folder:` Indicates a default draft folder for the **comp**, **dist**, **forw**, and **repl** commands. This information is stored in the MH profile. No default value is set.

`digest-issue-`*List*`:` Indicates to the **forw** command the last issue of the last volume sent for the digest *List*. This information is stored in the context file. No default value is set.

`digest-volume-`*List*`:` Indicates to the **forw** command the last volume sent for the digest *List*. This information is stored in the context file. No default value is set.

`MailDrop:` Indicates to the **inc** command your mail drop, if different from the default. This is superseded by the **$MAILDROP** environment variable. This information is stored in the MH profile. The default value is the **/usr/mail/$USER** file.

`Signature:` Indicates to the **inc** command your mail signature. This is superseded by the **$SIGNATURE** environment variable. This information is stored in the MH profile. No default value is set.

Profile Elements

The following profile elements are used whenever an MH program invokes another program. You can use the **.mh_profile** file to select alternate programs. The following list gives the default values:

`fileproc:`	**/usr/bin/refile**
`incproc:`	**/usr/bin/inc**
`installproc:`	**/usr/lib/mh/install-mh**
`lproc:`	**/usr/ucb/more**
`mailproc:`	**/usr/bin/mhmail**
`mhlproc:`	**/usr/lib/mh/mhl**

moreproc:	**/usr/ucb/more**
mshproc:	**/usr/bin/msh**
packproc:	**/usr/bin/packf**
postproc:	**/usr/lib/mh/spost**
rmmproc:	None
rmfproc:	**/usr/bin/rmf**
sendproc:	**/usr/bin/send**
showproc:	**/usr/ucb/more**
whatnowproc:	**/usr/bin/whatnow**
whomproc:	**/usr/bin/whom**

Environment Variables

Message Handler (MH) programs support the following environment variables:

$MH
Specifies a profile for an MH program to read. When you invoke an MH program, it reads the **.mh_profile** file by default. If you define the **$MH** environment variable, you can specify a different profile.

If the file of the **$MH** environment variable is not absolute (that is, it does not begin with a / (slash)), it is presumed to start in the current directory.

$MHCONTEXT
Specifies a context file that is different from the normal context file specified in the MH profile. If the value of the **$MHCONTEXT** environment variable is not absolute, it is presumed to start from your MH directory.

$MAILDROP
Indicates to the **inc** command the default mail drop. This supersedes the `MailDrop:` profile entry.

$SIGNATURE
Specifies your mail signature to the **send** and **post** commands. This supersedes the `Signature:` profile entry.

$HOME
Specifies your home directory to all MH programs.

$TERM, $TERMCAP
Specify your terminal type to the MH package. In particular, these environment variables tell the **scan** and **mhl** commands how to clear your terminal, and give the width and the length of your terminal in columns and lines respectively.

$editalt
Specifies an alternate message. This is set by the **dist** and **repl** commands during edit sessions so that you can read the distributed message or the answered message. This message is also available through a link called @ (at) sign in the current directory, if your current directory and the message folder are on the same AIX file system.

$mhdraft
Specifies the path of the working draft.

$mhfolder Specifies the folder containing the alternate message. This is set by the **dist** and **repl** commands during edit sessions, so you can read other messages in the current folder besides the one being distributed. The **$mhfolder** environment variable is also set by the **show**, **prev**, and **next** commands for use by the **mhl** command.

Example

The following example has the mandatory entry for `Path:`. The option `–alias` aliases is used when both the **send** command and the **ali** command are invoked. The **aliases** file resides in the mail directory. The message protection is set to `600`, which means that only the user has permission to read the message files. The signature is set to `Dan Carpenter`, and the default editor is `/usr/bin/vi`.

```
Path:          Mail
send:         –alias aliases
ali:          –alias aliases
Msg-Progtect: 600
Signature:     Dan Carpenter
Editor:        /usr/bin/vi
```

Implementation Specifics

This file is part of Message Handler in BOS Extensions 1.

Files

$HOME/.mh_profile	Contains the user profile.
UserMHDirectory/**context**	Contains the user context file.
Folder/**.mh_sequences**	Contains the public sequences for the folder specified by the *Folder* variable.

Related Information

The **chmod** command, **comp** command, **dist** command, **edit** command, **env** command, **folder** command, **forw** command, **inc** command, **install_mh** command, **mhl** command, **next** command, **post** command, **prev** command, **repl** command, **scan** command, **send** command, **show** command, **whatnow** command.

MH mhl.format File

Purpose

Controls the format of output for the **mhl** command.

Description

The **mhl.format** file controls the format of output when the **mhl** command is the message listing program. The full path of the **mhl.format** file is **/usr/lib/mh/mhl.format**. This file is the default file. The other files that must be specified if they are going to be used are:

- **mhl.digest** file
- **mhl.forward** file
- **mhl.reply** file.

Each line of the **mhl.format** file must have one of the following forms:

- ;*Comment*
- :*ClearText*
- *Component*:[*Variable*,...]
- *Variable*[*Variable*,...]

The content of these lines is described as follows:

; (semicolon) — A line beginning with a ; (semicolon) contains the comments specified by the *Comment* field that are ignored.

: (colon) — A line beginning with a : (colon) contains text for output (*ClearText*). A line that contains a : (colon) only produces a blank output line.

Component — A line beginning with the *Component* field defines the format of the specified component.

Variable — If the value specified by the *Variable* field follows a component, the variable applies only to that component. Lines having other formats define the global environment.

The entire **mhl.format** file is parsed before output processing begins. Thus, if the global setting of a variable setting is defined in multiple places, the last global definition for that variable describes the current global setting.

The following table lists the **mhl.format** file variables and their parameters:

File Variables for the mhl.format File		
Parameter	**Variable**	**Description**
Width	*integer*	Sets the screen width or component width.
Length	*integer*	Sets the screen length or component length.
OffSet	*integer*	Indents *Component* the specified number of columns.
OverflowText	*string*	Outputs *String* at the beginning of each overflow line.
OverflowOffset	*integer*	Indents overflow lines the specified number of columns.
CompWidth	*integer*	Indents component text the specified number of columns after the first line of output.
Uppercase	*flag*	Outputs text of *Component* in all uppercase characters.
NoUppercase	*flag*	Outputs text of *Component* in the case entered.
ClearScreen	*flag*/**G**	Clears the screen before each page.
NoClearScreen	*flag*/**G**	Does not clear the screen before each page.
Bell	*flag*/**G**	Produces an audible indicator at the end of each page.
NoBell	*flag*/**G**	Does not produce an audible indicator at the end of each page.
Component	*string*/**L**	Uses *String* as the name for the specified *Component* instead of the string *Component*.
NoComponent	*flag*	Does not output the string *Component* for the specified *Component*.
Center	*flag*	Centers *Component* on line. This variable works for one-line components only.
NoCenter	*flag*	Does not center *Component*.
LeftAdjust	*flag*	Strips off the leading whitespace characters from each line of text.
NoLeftAdjust	*flag*	Does not strip off the leading whitespace characters from each line of text.
Compress	*flag*	Changes new-line characters in text to space characters.
NoCompress	*flag*	Does not change new-line characters in text to space characters.
FormatField	*string*	Uses *String* as the format string for the specified *component*.
AddrField	*flag*	The specified *Component* contains addresses.
DateField	*flag*	The specified *Component* contains dates.
Ignore	*unquoted string*	Does not output component specified by *String*.

Variables that have integer or string values as parameters must be followed by an = (equal) sign and the integer or string value (for example, `overflowoffset=5`). String values must also be enclosed in double quotation marks (for example, `overflowtext="***"`). A parameter specified with the /**G** suffix has global scope. A parameter specified with the /**L** suffix has local scope.

Example

The following is an example of a line that could be displayed in the **mhl.format** file:

```
width=80,length=40,clearscreen,overflowtext="***".,overflowoffset=5
```

This format line defines the screen size to be `80` columns by `40` rows and specifies that the screen should be cleared before each page (`clearscreen`), that the overflow text should be flagged with the `***` string, and that the overflow indentation should be `5` columns.

Implementation Specifics

This file is part of Message Handler in BOS Extensions 1.

Related Information

The **ap** command, **dp** command, **mhl** command, **scan** command.

Understanding the Message Handler (MH) Command Output Format in *Communication Concepts and Procedures*.

MH mtstailor File

Purpose

Tailors the Message Handler (MH) environment to the local environment.

Description

The entries located in the **mtstailor** file specify how MH commands work. The following list describes the **mtstailor** file entries and their default values. All of the file entries are optional.

`localname:` Specifies the host name of the local system. If this entry is not defined, MH queries the system for the default value.

`systemname:` Specifies the host name of the local system in the UUCP domain. If this entry is not defined, MH queries the system for the default value.

`mmdfldir:` Specifies the location of mail drops. If this entry is present and empty, mail drops are located in the user's **$HOME** directory. If this entry does not exist, mail drops are located in the **/usr/mail** directory.

`mmdflfil:` Specifies the name of the file used as the mail drop. If this entry is not defined, the default file name is the same as the user name.

`mmdelim1:` Specifies the beginning-of-message delimiter for mail drops. The default value is the Ctrl + A key sequence four times followed by a new-line character (. 001. 001. 001. 001. 012). A Ctrl + A key sequence is a nonprintable character not displayed on the screen.

`mmdelim2:` Specifies the end-of-message delimiter for mail drops. The default value is the Ctrl + A key sequence four times followed by a new-line character (. 001. 001. 001. 001. 012). A Ctrl + A key sequence is a nonprintable character not displayed on the screen.

`mmailid:` Specifies whether support for the *MMailID* variable in the **/etc/passwd** file is enabled. If the `mmailid:` entry is set to a nonzero value, support is enabled. The `pw_gecos:` field in the **/etc/passwd** file has the following form:

`My Full Name` *MailID*

When support for the *MMailID* variable is enabled, the internal MH routines that deal with user and full names return the *MailID* variable and the `My Full Name` respectively. The default value is 0.

`lockstyle:` Specifies the locking discipline. A value of 1 (one) creates lock names by appending `.lock` to the name of the file being locked. The default value is 1.

`lockldir:` Specifies the directory for locked files. The default value is the **/etc/locks** file.

`sendmail:` Specifies the path name of the **sendmail** command. The default value is the **/usr/lib/sendmail** file.

`maildelivery:` Specifies the path name of the file containing the system default mail delivery instructions. The default value is the **/usr/lib/mh/maildelivery** file.

`everyone:` Specifies the users to receive messages addressed to everyone. All users having UIDs greater than the specified number (not inclusive) receive messages addressed to everyone. The default value is 200.

Implementation Specifics

This file is part of Message Handler in BOS Extensions 1.

File

/usr/lib/mh/mtstailor Contains MH command definitions.

Related Information

The **.maildelivery** file.

NFS bootparams File

Purpose

Contains the list of client entries that diskless clients use for booting.

Description

Contains a list of client entries that diskless clients use for booting. The first item of each entry is the name of the diskless client. Each entry should contain the following information:

- Name of client
- List of keys, names of servers, and path names

Items are separated by tab characters.

Examples

The following is an example of a **/etc/bootparams** file:

```
myclient root=myserver:/nfsroot/myclient \
  swap=myserver:/nfsswar/myclient \
  dump=myserver:/nfsdump/myclient
```

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

NFS exports File

Purpose

Contains a list of directories that can be exported to Network File System (NFS) clients.

Description

The **/etc/exports** file contains an entry for each directory that can be exported to NFS clients. This file is read automatically by the **exportfs** command. If you change this file, you must run the **exportfs** command before the changes can affect the way the daemon operates.

Only when this file is present during system startup does the **rc.nfs** script execute the **exportfs** command and start the **nfsd** and **mountd** daemon.

Note: You cannot export either a parent directory or a subdirectory of an exported directory that is within the same file system.

Entries in the file are formatted as follows:

Directory *–Option* [, *Option*] ...

These fields are defined as follows:

Directory Pathname of the directory

Option Specifies optional characteristics for the directory being exported. You can enter more than one option by separating them with commas. Choose from the following options:

ro Exports the directory with read-only permission. Otherwise, if not specified, the directory is exported with read-write permission.

rw = *Client* [:*Client*]
Exports the directory with read-write permission to the machines specified by the *Client* parameter and read-only to all others. The *Client* parameter can be either the hostname or the network name. If a *rw* host name is not specified, the directory is exported with read-write permission to all.

anon= *UID* If a request comes from a root user, use the *UID* value as the effective user ID.

The default value for this option is –2. Setting the value of the *anon* option to –1 disables anonymous access. Note that, by default, secure NFS accepts nonsecure requests as anonymous, and users who want more security can disable this feature by setting *anon* to a value of –1.

root = *HostName*[:*HostName,...*]
Gives root access only to the root users from the specified *HostName*. The default is for no hosts to be granted root access.

access = *Client*[:*Client*,...]
: Gives mount access to each client listed. A client can be either a host name or a netgroup name. Each client in the list is first checked for in the **/etc/netgroup** database and then in the **/etc/hosts** database. The default value allows any machine to mount the given directory.

secure
: Requires clients to use a more secure protocol when accessing the directory.

A # (pound sign) anywhere in the file indicates a comment that extends to the end of the line.

Examples

1. Export to netgroup clients:

   ```
   /usr -access=clients
   ```

2. Export to the world:

   ```
   /usr/local
   ```

3. Export to only these machines:

   ```
   /usr2 -access=hermes:zip:tutorial
   ```

4. Give Root access only to these:

   ```
   /usr/tps -root=hermes:zip
   ```

5. Convert client root users to guest UID=100:

   ```
   /usr/new -anon=100
   ```

6. Export read-only to everyone:

   ```
   /usr/bin -ro
   ```

7. Allow several options on one line:

   ```
   /usr/stuff -access=zip,anon=-3,ro
   ```

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

Files

/etc/xtab	Lists currently exported directories.
/etc/hosts	Contains an entry for each host on the network.
/etc/netgroup	Contains information about each user group on the network.

Related Information

The **exportfs** command.

The **nfsd** daemon.

How to Export a Directory Using NFS in *Communication Concepts and Procedures*.

NFS networks File

Purpose

Contains information about networks on the Internet network.

Description

The **/etc/networks** file contains information regarding the known networks that make up the Internet. The file has an entry for each network. Each entry consists of a single line with the following information:

- Official network name
- Network number
- Aliases.

Items are separated by any number of blanks and/or tab characters. A # (pound sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Implementation Specifics

This file is not supported by AIX. However, if this file resides on your system, the network information service (NIS) software will create a map for it.

NFS rpc File

Purpose

A database for **rpc** program numbers

Description

The **/etc/rpc** file contains user-readable names that can be used in place of **rpc** program numbers. Each line of the file contains the following fields:

Name of Server for the RPC Program *RPC Program Number* *Aliases*

These fields are defined as follows:

Name of Server for the RPC Program
This field lists the name of the server daemon that provides the RPC program.

RPC Program Number
The number assigned to the program by the RPC protocol.

Aliases This is a list of alternate names by which the service might be requested.

The fields are separated by any number of blanks or tab characters. A # (pound sign) indicates the beginning of a comment. Characters up to the end of the commented line are not interpreted by routines which search the file.

Example

Here is an example of an **/etc/rpc** file:

```
portmapper      100000      portmap sunrpc
rstatd          100001      rstat rup perfmeter
rusersd         100002      rusers
nfs             100003      nfsprog
ypserv          100004      ypprog
mountd          100005      mount showmount
```

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

NFS xtab File

Purpose

Contains entries for directories that are *currently* exported.

Description

The **/etc/xtab** file contains entries for directories that are currently exported. This file should only be accessed by programs using the **getexportent** subroutine. To remove entries from this file, use the **–u** option of the **exportfs** command.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

Files

/etc/exports	Lists the directories that the server can export.
/etc/hosts	Contains an entry for each host on the network.
/etc/netgroup	Contains information about each user group on the network.

Related Information

The **exportfs** command.

How to Export a Directory Using NFS in *Communication Concepts and Procedures*.

NIS ethers File

Purpose

Contains the Ethernet addresses of hosts on the Internet network.

Description

The **/etc/ethers** file contains information regarding the known (48-bit) Ethernet addresses of hosts on the Internet. The file contains an entry for each host. Each entry consists of the following information:

- Ethernet address
- Official host name

Items are separated by any number of blanks and/or tab characters. A # (pound sign) indicates the beginning of a comment that extends to the end of the line.

The standard form for Ethernet addresses is `x:x:x:x:x:x:` where `x` is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than a space, tab, newline, or comment character. It is intended that host names in the **/etc/ethers** file correspond to the host names in the **/etc/hosts** file.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

Related Information

The **/etc/hosts** file.

NIS netgroup File

Purpose

Lists the groups of users on the network.

Description

The **/etc/netgroup** file defines network-wide groups. This file is used for checking permissions when doing remote mounts, remote logins, and remote shells. For remote mounts, the information in the **netgroup** file is used to classify machines. For remote logins and remote shells, the file is used to classify users. Each line of the **netgroup** file defines a group and is formatted as follows:

Groupname Member1 Member2 ...

where *Member* is either another group name, or consists of three entries as follows:

hostname, username, domainname

Any of these three fields can be empty, in which case it signifies a wild card. Thus *universal (, ,)* defines a group to which everyone belongs.

Field names that begin with something other than a letter, digit or underscore (such as –) work in precisely the opposite fashion. For example, consider the following entries:

```
justmachines     (analytica,-,ibm)
justpeople       (-,babbage,ibm)
```

The machine `analytica` belongs to the group `justmachines` in the domain `ibm`, but no users belong to it. Similarly, the user `babbage` belongs to the group `justpeople` in the domain `ibm`, but no machines belong to it.

A gateway machine should be listed under all possible host-names by which it may be recognized: *wan (gateway , ,) (gateway-ebb , ,)*

The *domainname* field refers to the domain *n* in which the triple is valid, not the name containing the trusted host.

Example

The following is an excerpt from a **netgroup** file:

```
machines  (venus, -, star)
people    (-, bob, star)
```

In this example, the machine named `venus` belongs to the group `machines` in the `star` domain. Similarly, the user `bob` belongs to the group `people` in the `star` domain.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

Related Information

The **makedbm** command.

The **ypserv** daemon.

How to Mount a File System Using NFS and Maintaining an NFS Client in *Communication Concepts and Procedures.*

NIS netmasks File

Purpose

Contains network masks used to implement Internet Protocol (IP) standard subnetting.

Description

The **/etc/netmasks** file contains network masks used to implement IP standard subnetting. This file contains a line for each network that is subnetted. Each line consists of the network number, any number of spaces or tabs, and the network mask to use on that network. Network numbers and masks may be specified in the conventional IP '.' (dot) notation (similar to IP host addresses, but with zeroes for the host part). The following number is a line from a **netmask** file:

128.32.0.0 255.255.255.0

This number specifies that the Class B network `128.32.0.0` has 8 bits of subnet field and eight bits of host field, in addition to the standard 16 bits in the network field. When running network information service, this file on the master is used for the **netmasks.byaddr** map.

Implementation Specifics

This file is not supported by AIX. However, if this file resides on your system, NIS will create a map for it.

NIS publickey File

Purpose

Contains public or secret keys for NIS maps.

Description

The **/etc/publickey** file is the public key file used for secure networking. Each entry in the file consists of a network user name (which may either refer to a user or a host name), followed by the user's public key (in hex notation), a colon, and then the user's secret key encrypted with its login password (also in hex notation).

This file is altered either by the user through the **chkey** command or by the person who administers the system through the **newkey** command. The **/etc/publickey** file should only contain data on the NIS master server, where it is converted into the **publickey.byname** NIS map.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

Related Information

The **chkey** command, **keylogin** command, **newkey** command.

The **keyserv** daemon, **ypupdated** daemon.

How to Use the NFS Secure Option When Exporting a Directory and How to Use the NFS Secure Option When Mounting a File System in *Communication Concepts and Procedures.*

NIS updaters File

Purpose

A makefile for updating NIS maps.

Description

The **/etc/yp/updaters** file is a makefile used for updating NIS maps. NIS maps can only be updated in a secure network; that is, one that has a **publickey** file. Each entry in the file is a make target for a particular NIS map. For example, if there is an NIS map named `passwd.byname` that can be updated, there should be a make target named `passwd.byname` in the **updaters** file with the command to update the file.

The information necessary to make the update is passed to the **update** command through standard input. The information passed is described below (all items are followed by a newline except for actual bytes of key and actual bytes of data.)

- Network name of client wishing to make the update (a string)
- Kind of update (an integer)
- Number of bytes in key (an integer)
- Actual bytes of key
- Number of bytes in data (an integer)
- Actual bytes of data.

After getting this information through standard input, the command to update the map decides whether the user is allowed to make the change. If the user is not allowed, the command exits with the YPERR_ACCESS status. If the user is allowed to make the change, the command should make the change and exit with a status of zero. If there are any errors that may prevent the **updaters** file from making the change, it should exit with the status that matches a valid NIS error code described in the **<rpcsvc/ypclnt.h>** file.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System Runtime.

Related Information

The **update** command.

The **ypupdated** daemon.

Administering Secure NFS in *Communication Concepts and Procedures.*

Network Information Service (NIS) Overview for System Management in *Communication Concepts and Procedures.*

SNMP smpl.pwinput File

Purpose

Provides sample input to the **mksnmppw** command.

Description

The **smpl.pwinput** file is a sample input file to the **mksnmppw** command, which creates the **/etc/snmpd.pw** file. The **snmp.pw** file may be changed while the **snmpd** daemon is running. The changes will take effect immediately.

This file contains records in the following format:

Community IP_address Address_mask XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

where:

Community Is a 1–to–8 character community name.

Ip_address Is an IP address in the dotted decimal format.

Address_mask Is a mask in dotted decimal format.

XX...XX Is a 32-character sequence of attributes. An attribute is enabled by the character s, and is disabled by the character –.

Each SNMP request submitted to the SNMP daemon contains the IP address of the originator and a community name. The records from the **/etc/snmpd.pw** file are checked until the first record is found whose IP address and community name match those of the request. The request is assigned the attributes enabled in the 32-character attribute array. The address mask determines which part of the IP addresses are compared.

Attributes

The 32-character attribute array is used as follows (characters are numbered left to right):

Character 1: s Enables tracing.

Character 2: s Enables reply tracing, if character 1 is s.

Character 3: s Enables request tracing, if character 1 is s.

Characters 4 through 30
Unused. They *must* be –, except for the special entry (snmpdprivate) as illustrated below.

Character 31: s Enables set request processing.

Character 32: s Enables request processing.

Character 32: – Rejects all requests from the IP address, under the given community name.

Special Entry

The **/etc/snmpd.pw** file allows one special entry:

```
snmpdprivate    0.0.0.0          255.255.255.255
        ——xx————————————————————————
```

where:

Character 4: s — Enables tracing of all external changes, such as MIB traps.

Character 5: s — Enables tracing of internal errors.

Other character positions *must* be –.

All tracing attributes may be changed while the SNMP daemon is running by modifying the **snmp.pwinput** file and running the **mksnmppw** command. For each request processed, the **/etc/snmpd.pw** file is checked to see if it has changed. At this time, the new trace attributes will be processed.

Note: The **snmp.pwinput** file should *not* be generally accessible since it is not encrypted and contains information for authentication. It is recommended that read permission be granted only to members of the system group.

Examples

1. Example of entries in the **smpl.pwinput** file:

```
monitor         128.0.0.4       255.255.255.255
        s-s--------------------------ss
monitor         129.1.2.5       255.255.255.0
        ss-----------------------------s
monitor         127.0.0.1       255.255.255.255
        ss-----------------------------s-
```

   In this example, the first entry will have tracing enabled and will generate trace information for all replies sent to IP address `128.0.0.4`. The agent will also process SET requests sent to it from this address if the community name specified for this address is `monitor`.

   The second entry will have tracing enabled and will trace information for all requests from all IP addresses that start with `129.1.2.`, as indicated by the mask. GET and GET–NEXT requests from network `129.1.2` will be accepted, but SET requests will not.

   The third entry causes all requests from host `127.0.0.1` to be ignored because character 32 is not –.

2. In the following example, all requests under community name `monitor` will be accepted regardless of the IP address.

```
monitor         0.0.0.0         0.0.0.0
        sss----------------------------s
```

3. Example of a special entry in the **smpl.pwinput** file:

```
snmpdprivate    0.0.0.0         255.255.255.255
        ---ss-------------------------
```

Implementation Specifics

This file is part of Simple Network Management Protocol Agent Applications in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Related Information

The **mksnmppw** command.

The **snmpd** command.

Understanding the Simple Network Management Protocol (SNMP), Understanding the SNMP Daemon in *Communications Programming Concepts*.

SNMP snmptrap.dest File

Purpose

Lists the hosts and community names that are to receive trap messages.

Description

The **/etc/snmptrap.dest** file is required if SNMP trap messages are to be sent. The SNMP daemon will run if the **/etc/snmptrap.dest** file does not exist, and trap events will be logged if tracing is enabled, but trap messages will not be transmitted.

The **/etc/snmptrap.dest** file contains a record for each host that receives trap data. The record has the format indicated by the following examples:

```
# HOST    PROTOCOL    COMMUNITY    FLAGS
host1     udp         monitor
host2     udp         test         fcwud
```

The first field is the host name to which the trap message should be sent. The second field is the protocol. Only UDP is currently supported; TCP is accepted but ignored. The third field is the community name to use in the message. The last field specifies which trap types the corresponding hosts wish to receive. The flags are specified as single characters:

a authenticationFailure trap

c coldStart trap

d linkDown trap

e enterpriseSpecific (unsupported, but may be specified)

f force coldStart after linkUp

n egpNeighborLoss trap

u linkUp trap

w warmStart trap.

If no flags are specified, "acdenuw" is assumed.

Lines that begin with # are treated as comments.

In the example above, `host1` receives all traps except the forced coldStart trap accompanying a linkUp trap. A forced coldStart trap will be sent to `host2`, but the authenticationFailure and egpNeighborLoss traps will not be sent. The forced coldStart trap may be used to force a host to reset the time intervals for its statistical calculations, in case it does not do so in response to the linkUp trap. A linkUp trap may indicate that MIB variable values associated with the interface have been reset.

Since the **/etc/snmptrap.dest** file contains community names in an unencrypted format, read permission should be granted only to users with root user privileges and to members of the system group.

Implementation Specifics

This file is part of Simple Network Management Protocol Agent Applications in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Related Information

The **snmpd** command.

Understanding the Simple Network Management Protocol (SNMP), Understanding the SNMP Daemon in *Communications Programming Concepts*.

TCP/IP rc.tcpip File

Purpose

Initializes daemons at each system restart.

Description

The **/etc/rc.tcpip** file is a shell script that, when executed, uses SRC commands to initialize selected daemons. The **rc.tcpip** shell script is automatically executed with each system restart. It can also be executed at any time from the command line.

Most of the daemons that can be initialized by the **rc.tcpip** file are specific to TCP/IP. These daemons are:

- **inetd** (started by default)
- **gated**
- **routed**
- **named**
- **timed**
- **rwhod**

Note: Running the **gated** and **routed** daemons at the same time on a host may cause unpredictable results.

There are also daemons specific to the base operating system or to other applications that can be started through the **rc.tcpip** file. These daemons are:

- **lpd**
- **portmap**
- **sendmail**
- **syslogd**

The **syslogd** daemon is started by default.

Examples

1. The following stanza starts the **syslogd** daemon.

```
#Start up syslog daemon (for error and event logging)
if [-f /etc/syslogd]; then
startsrc -f syslogd
```

2. The following stanza starts the **routed** daemon, but not the **gated** daemon.

```
#Start up routing daemons (only start one)
if [-f /etc/routed]; then
        startsrc -s routed -a "-q"
#fi
#ip [-f /etc/gated]; then
#       startsrc -s gated
#fi
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Related Information

The **startsrc** command, **stopsrc** command.

The **inetd** daemon, **gated** daemon, **lpd** daemon, **named** daemon, **portmap** daemon, **routed** daemon, **rwhod** daemon, **sendmail** daemon, **syslogd** daemon, **timed** daemon.

Chapter 2. File Formats

File Formats Overview

Certain files in the AIX operating system are required to have a specific format. The formats of the files that are provided with the operating system are discussed in the documentation for those files. If a file is generated by either the system or a user rather than provided on the distribution medium, it is discussed as a *file format* in this documentation. File formats often are also associated with header files that contain C-language definitions and structures for the files.

More information about the following file formats is provided in this documentation:

acct Describes the format of the records in the system accounting files.

a.out Describes the structure of the standard **a.out** file and its associated header files.

ar Describes the format of an archive file.

audit Describes values used by the auditing system as well as the structure of a bin.

core Describes the structures created in a **core** file as a result of a core dump.

cpio Describes the **cpio** (copy in/out) archive file

EM78 Customization
Specifies the keyboard layout, screen colors, and field attribute modes to be used in 3278/79 Terminal Emulation.

MH Alias Defines aliases for the Message Handler (MH).

nterm Describes the format of the terminal driver tables for the **nroff** command.

PC Simulator Startup
Specifies options when starting PC Simulator.

profile Describes the format of the **profile** and **.profile** files, which set the user environment at login time.

sccsfile Describes the format of a Source Code Control System (SCCS) file.

troff Describes the output language of the **troff** command.

troff Font Describes the format of the **troff** command font files.

utmp, wtmp, failedlogin
Describes the format of the user and accounting information in the **utmp**, **wtmp**, and **failedlogin** files.

ATE File Formats

ate.def Determines default settings for use in asynchronous connections and file transfers.

ATE Dialing Directory
Lists phone numbers that the ATE program uses to establish modem connections.

BNU File Formats

Devices Contains information about devices on the local system that can establish a connection to a remote computer using the Basic Networking Utilities (BNU) program.

Dialcodes Contains the initial digits of telephone numbers used to establish remote connections over a phone line.

Dialers Lists modems used for Basic Networking Utilities (BNU) remote communications links.

Maxuuscheds Limits the number of instances of the **uusched** and **uucico** daemons that can run simultaneously.

Maxuuxqts Limits the number of instances of the BNU **uuxqt** daemon that can run simultaneously on the local system.

Permissions Specifies BNU permissions for remote systems that call or are called by the local system.

Poll Specifies when the BNU program should poll remote systems.

Systems Lists remote computers with which users of the local system can communicate using the Basic Networking Utilities (BNU) program.

HCON File Formats

e789_ctbl.p Source for the default binary color definition table

e789_ktbl.p Source for the default binary keyboard definition table

func_names Keyboard function names

keynames Key names

nls_names Color and attribute names.

tip File Formats

phones Describes connections used by the **tip** command to contact remote systems.

remote Describes remote systems contacted by the **tip** command.

.tiprc Provides initial settings of variables for the **tip** command.

TCP/IP System Management File Formats

.3270keys Defines a user keyboard mapping and colors for TELNET (3270).

Domain Cache Defines the root name server or servers for a DOMAIN name server host.

Domain Data file format
Stores name resolution information for the **named** daemon.

Domain Local Data
Defines the local loopback information for **named** on the name server host.

Domain Reverse Data
Stores reverse name resolution information for the **named** daemon.

ftpusers Specifies local user names that cannot be used by remote FTP clients.

gated.conf Contains configuration information for the **gated** daemon.

gateways Specifies Internet routing information to the **routed** and **gated** daemons on a network.

hosts Defines the Internet Protocol (IP) name and address of the local host and specifies the names and addresses of remote hosts.

hosts.equiv Specifies remote systems that can execute commands on the local system.

hosts.lpd Specifies remote hosts that can print on the local host.

inetd.conf Defines how the **inetd** daemon handles Internet service requests.

named.boot Defines how **named** initializes the DOMAIN name server file.

.netrc Specifies automatic login information for the **ftp** and **rexec** commands.

networks Contains the network name file.

protocols Defines the Internet protocols used on the local host.

rc.net Defines host configuration for the following areas: network interfaces, host name, default gateway, and any static routes.

resolv.conf Defines DOMAIN name server information for local resolver routines.

.rhosts Specifies remote users that can use a local user account on a network.

services Defines the sockets and protocols used for Internet services.

Standard Resource Record Format
Defines the format of lines in the DOMAIN data files.

Related Information

The Header Files Overview, which describes header files in general and lists header files discussed in this documentation.

The Special Files Overview, which defines and describes special files in general and lists special files discussed in this documentation.

acct File Format

Purpose

Provides the accounting file format for each process.

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 **acct.h** file defines the records in these files, which are written when a process exits.

acct Structure

The **acct** structure in the **acct.h** header file contains the following fields:

ac_flag — An accounting flag for the process for which the accounting record is written:

- AFORK — The process was created using a **fork** command but an **exec** subroutine has not yet followed. The **exec** subroutine turns off the AFORK flag.
- ASU — The process used root user authority.

ac_stat — Exit status. A flag that indicates how the process terminated.

ac_uid — The user ID of the process for which the accounting record is written.

ac_gid — The group ID of the process for which the accounting record is written.

ac_tty — The terminal from which the process was started.

ac_btime — Beginning time. The time at which the process started.

ac_utime — The amount of user time (in clock ticks) used by the process.

ac_stime — The amount of system time (in clock ticks) used by the process.

ac_etime — The amount of time (in clock ticks) elapsed since the command ran.

ac_mem — The average amount of memory used by the process. 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* plus *text size*) divided by the *number of in-memory processes using text*.

ac_io — The number of characters transferred by the process.

ac_rw — The number of blocks read or written by the process.

ac_comm — The name of the command that started the process. A child process created by a **fork** subroutine receives this information from the parent process. An **exec** subroutine resets this field.

tacct Structure

The **tacct** structure, which is not part of the **acct.h** header file, represents the total accounting format used by the various accounting commands:

```
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 */
    };
```

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/sys/acct.h The path to the **acct.h** header file.

Related Information

The **acct** subroutine, **fork** subroutine, **exec** subroutine.

The **utmp.h** file.

The **acct/*** command, **acctcms** command, **acctcom** command, **acctcon** command, **acctdisk** command, **acctmerg** command, **acctprc** command, **runacct** command.

a.out File Format

Purpose

Provides common assembler and link editor output.

Description

The **as** (assembler), compilers, and **ld** (link editor) programs produce an output file (the **a.out** file by default) in the following format. The **a.out** file, which is runnable if no unresolved external references or errors are found in the source, can contain the following sections:

- A file header
- An auxiliary header
- Section headers for each of the file's raw data sections
- Raw data sections
- Relocation information for each raw data section
- Line number information for each raw data section
- Special data sections
- A symbol table section
- A string table (if long symbols are used).

Not every **a.out** file contains every section. In particular, the line number, symbol table, and string table sections are omitted if the program was linked with the **–s** flag of the **ld** command or if these sections were removed by the **strip** command.

Additionally, a loader section is required in order to load dynamically an **a.out** file into memory for operation. Loading an **a.out** file into memory creates three logical segments: the text segment, the data segment (initialized data followed by data that is initialized to all zeros), and a stack segment. The figure on the following page shows the format of an **a.out** file.

The **a.out.h** header file defines the structure of the standard **a.out** file. The **a.out.h** header file includes the following header files as object file components:

- **/usr/include/xcoff.h**
- **/usr/include/filehdr.h**
- **/usr/include/aouthdr.h**
- **/usr/include/scnhdr.h**
- **/usr/include/loader.h**
- **/usr/include/typchk.h**
- **/usr/include/exceptab.h**
- **/usr/include/dbug.h**
- **/usr/include/reloc.h**
- **/usr/include/linenum.h**

- **/usr/include/syms.h**
- **/usr/include/storclass.h**
- **/usr/include/dbxstclass.h.**

Composite Header
File header (fixed part) Optional auxilliary header (extension) Section headers: .text, .data, .bss, .pad .loader .debug, .typchk, .except
Sections
Raw Data
Executables and alignment padding: .text, .data, .pad sections Loader section: .loader Import file IDs Import and export symbols Loader and export symbols Symbolic debugger (stab) section Linkage editor (type check) section Traceback (exception) section
Relocation Data
Linkage editor relocation data for .text and .data sections
Line Number Data
Symbolic debugger (dbx) line number data for .text and .data sections
Symbol Table
Master symbol table for linkage editor and symbolic debugger symbols Contains .file, external, and local symbol definitions
String Table
String table for long symbol names

Figure 2–1. a.out File Format

Access Routines for the a.out File

To ease program access to the XCOFF file sections, use the access routines in the **libld** library. This library contains routines to open, close, and access sections of an XCOFF file.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

File Header Section of the a.out File

f_magic — Magic number. Target machine and environment on which the object file is executable. The value equals 0x01DF (0737).

Targets are defined in the **filehdr.h** header file as well as in standard COFF definitions.

f_nscns — The number of section headers (and sections) contained in the file. The first section is section number one.

f_timdat — The time and date stamp that indicates when the file was created (number of elapsed seconds since 00:00:00 GMT, January1, 1970).

f_symptr — A file pointer (offset from the beginning of the file) to the start of the symbol table.

f_nsyms — The number of entries in the symbol table. Each entry is 18 bytes in length.

f_opthdr — The size (in bytes) of the optional auxiliary header.

f_flags — Flags that describe the type of object file:

F_RELFLG	0x0001	The relocation information is stripped from the file. This flag is not set if the relocation information is not required.
F_EXEC	0x0002	The file is executable; there are no unresolved external references.
F_LNNO	0x0004	The line numbers are stripped from the file.
F_LSYMS	0x0008	The local symbols are stripped from the file.
F_MINMAL	0x0010	Reserved.
F_UPDATE	0x0020	Reserved.
F_SWABD	0x0040	Reserved.
F_AR16WR	0x0080	The byte ordering of a 16-bit reversed (Personal Computer AT) architecture.
F_AR32WR	0x0100	The byte ordering of a 32-bit reversed (PS/2) architecture.
F_AR32W	0x0200	The byte ordering of a 32-bit word (RISC System/6000) architecture.
F_PATCH	0x0400	Reserved.
F_DYNLOAD	0x1000	The file ID is dynamically loadable and executable. The external references are resolved by way of imports and might contain exports and loader relocation.
F_SHROBJ	0x2000	The file ID is a shared object (shared library). The file is separately loadable, it is not normally bound with other objects, and loader symbols are used as automatic exports symbols.

Optional Auxiliary Header for the a.out File

The optional header contains system-dependent information. By convention, the first 28 bytes of this section contain standard **a.out** header information. In this use, the system-dependent information follows the standard information and serves to minimize how much of the file must be processed by the system loader when it performs a relocating load operation. The link editor always produces an optional header; however, this is not required of the compilers and assemblers. The **aouthdr.h** file, which defines the optional header, contains the following fields:

o_mflag	(magic) Indicates to the operating system how the file should be executed. A value of 0x010B in this field indicates that the text and the data are aligned in the file and can be paged.
o_vstamp	(vstamp) The format version identifier for this option header. The value 0x0001 is the current value assigned.
o_tsize	(tsize) The size (in bytes) of the .text raw data section. This field typically contains the read-only executable instructions of the program. Some implementations require the .text to be padded to a fullword in length.
o_dsize	(dsize) The size (in bytes) of the .data raw data section. This field contains the initialized data of the program and is classified as read or write. Some implementations require the .data to be padded to a fullword in length.
o_bsize	(bsize) The size (in bytes) of the .bss data, which defines the length of uninitialized data of the program and is classified as read or write.
o_entry	(entry) The entry point function descriptor address as assigned by the linkage editor. The function descriptor contains the address of both the entry point and the TOC for the object file.
o_text_start	(text_start) Base address of .text raw data. This represents the linkage editor origin address value assigned (used) for the first byte of the .text raw data. This is the same value as contained in the s_vaddr field of the section header for the .text section.
o_data_start	(data_start) Base address of .data raw data. This represents the linkage editor origin address value assigned (used) for the first byte of the .data raw data. This is the same value as contained in the s_vaddr field of the section header for the .data section. The uninitialized data (.bss) is considered to be contiguous to the .data.

The following set of definitions are the extensions for AIX Version 3:

o_toc	The TOC address as assigned by the linkage editor. This field is not used by the AIX Version 3 loader. It is determined from the entry point descriptor.
o_snentry	The number of the file section that contains the entry point descriptor.
o_sntext	The number of the file section that contains the text.
o_sndata	The number of the file section that contains the data.
o_sntoc	The number of the file section that contains the TOC.
o_snloader	The number of the file section that contains the dynamic loader information.
o_snbss	The number of the file section that contains the .bss data.
o_algntext	Log 2 of the max alignment relocation factor for the .text section.

o_algndata — Log 2 of the max alignment relocation factor for the .data section.

o_modtype — Module type field. One of the following 2-byte ASCII character strings:

- 1L — Single use. The module requires distinct copies of the data section for each load.
- RE — Reusable. The module requires a private copy of the data area for each process that is dependent on the module.
- RO — Read-only. The module is read-only and can be used by multiple processes at the same time.

o_resv1 — Reserved.

o_maxstack — The maximum stack size allowed for this object. If this field is zero, then the system default maximum stack size is used.

o_resv2 — Reserved.

Section Headers for the a.out File

Each raw data section of the XCOFF file has a corresponding section header. The size and format is that of a standard COFF file. The **scnhdr.h** file, which defines the section headers, contains the following fields:

s_name — An 8-character, null-padded section name.

s_paddr — The physical address of the section. This is the address used by the compiler or the linkage editor to represent the address of the first byte in the section.

s_vaddr — The virtual address of the section. This value is the same as the s_paddr field.

s_size — The size (in bytes) of the raw data.

s_scnptr — A file pointer to the raw data for the section. The file pointer is relative to the first byte of the file header.

s_relptr — A file pointer to the relocation entries for the section. The file pointer is relative to the first byte of the file header.

s_lnnoptr — A file pointer to the line number entries for the section. The file pointer is relative to the first byte of the file header.

s_nreloc — The number of relocation entries for the section.

s_nlnno — The number of line number entries for the section.

s_flags — The COFF flags that define the section type. The low–order pair of bytes are used. Valid AIX COFF bit values are:

0x0000	STYP_REG	Regular section.
0x0001	STYP_DSECT	Reserved.
0x0002	STYP_NOLOAD	Reserved.
0x0004	STYP_GROUP	Reserved.
0x0008	STYP_PAD	Pad section.
0x0010	STYP_COPY	Reserved.
0x0020	STYP_TEXT	Executable text (code).

0x0040	STYP_DATA	Initialized data or TOC.
0x0080	STYP_BSS	Uninitialized data.
0x0100	STYP_EXCEPT	Exception section (for exception reason decode).
0x0200	STYP_INFO	Reserved.
0x0400	STYP_OVER	Reserved.
0x0800	STYP_LIB	Reserved.
0x1000	STYP_LOADER	Loader section (imports, exports, relocation).
0x2000	STYP_DEBUG	Debug section (symbolic debugger stab string section).
0x4000	STYP_TYPCHK	Type check section (parameter type check string section).
0x8000	STYP_OVRFLO	Relocation or line number field overflow section.

Programming Usage Notes

- Use the s_flags field instead of the s_name field to determine a section type.
- The following conventions are used if there are more than 65,534 relocation or line number entries in a section.
  - A section header that has its s_flags field set to STYP_TEXT or STYP_DATA puts the value of 0xFFFF in both the s_nreloc and s_nlnno fields to indicate that an auxiliary section header contains the relocation and line number count information in the s_paddr and s_vaddr fields, respectively.
  - A section header that has its s_flags field set to STYP_OVRFLO is regarded as an auxiliary section header to be used in conjunction with another primary section header. The auxiliary section header is used to handle a section that requires more than 65,534 relocation entries or line number entries. The following fields take on special meanings in an auxiliary section header:

    s_paddr — The number of relocation entries actually required by the primary section. This field is used as a replacement for the s_nreloc field.

    s_vaddr — The number of line number entries actually required by the primary section. This field is used as a replacement for the s_nlnno field.

    s_nreloc — The file section number of the primary section header.

    s_nlnno — The file section number of the primary section header.

    The s_size and s_scnptr fields are not used in an auxiliary section header; the s_name, s_relptr, s_lnnoptr, and s_flags fields have the same meanings as in a primary section header.

Raw Data Sections for the a.out File

The data sections of an XCOFF file follow the composite header portion of the file. A data section has three independent parts:

- A raw data part
- An optional relocation data part
- An optional line number table part.

Each section header provides the file position information for its associated section.

An XCOFF file has the following sections defined:

- The .text, .data, and .bss sections contain the executable object code. The relocation parts associated with the .text and .data sections contain the full linkage editor relocation information so it can be used for replacement link editing. Only the .text section contains a line number part. The parts associated with the executable code are produced by the compilers and assemblers.
- The .pad section is a null-filled raw data section that is used to align a subsequent section in the file on some defined boundary such as a file block boundary or a system page boundary.
- The .loader section contains the dynamic loader information in its raw data portion. This section is generated by the linkage editor and has its own self–contained symbol table and relocation table.
- The .debug section is defined to contain the stab (symbol table) or dictionary information required by the symbolic debugger.
- The .typchk section is defined to contain parameter and argument type-checking strings.
- The .except section is defined to contain the exception tables used to identify the reasons for an exception in program execution.

All of the information for the .loader, .debug, .typchk, and .except sections is contained in the raw data part. There is no relocation part or line number part in these sections. The .debug, .typchk, and .except sections are produced by compilers and assemblers. References to items in these sections are made from the symbol table.

Relocation Information for the a.out File

For AIX Version 3, only the .text and .data sections have relocation requirements. The linkage editor will generate any necessary relocation entries in the .loader section that will be required by the system dynamic loader.

The **reloc.h** file, which defines the structure for a relocation data entry, contains the following fields:

r_vaddr — The virtual address of the relocatable reference (adcon) in the raw data that requires modification by the linkage editor.

r_symndx — A zero-based index into the symbol table of the referenced symbol.

r_type — Relocation type. The structures that define the types of relocation to be performed on the relocation entry are as follows:

- r_sign — A 1-bit field that indicates whether the relocatable reference is signed (1) or unsigned (0).
  A 2-bit field that is reserved.
- r_len — A 5-bit field that contains the bit length minus 1 of the relocatable reference.

r_rtype — An 8-bit relocation type field that indicates to the linkage editor which relocation algorithm to use for the relocatable reference. The following relocation types are defined:

0x00	R_POS	A(sym) Positive relocation.
0x01	R_NEG	–A(sym) Negative relocation.
0x02	R_REL	A(sym–*) Relative to self.
0x03	R_TOC	A(sym–TOC) Relative to TOC.
0x12	R_TRL	A(sym–TOC) TOC Relative indirect load (modifiable L to LA).
0x13	R_TRLA	A(sym–TOC) TOC Relative load address (modifiable LA to L).
0x05	R_GL	A(External TOC of sym) Global linkage – external TOC address.
0x06	R_TCL	A(Local TOC of sym) Local object TOC address.
0x0C	R_RL	A(sym) Positive indirect load (modifiable L to LA).
0x0D	R_RLA	A(sym) Positive load address (modifiable LA to L).
0x0F	R_REF	AL0(sym) Non-relocating reference to prevent garbage collection of a symbol.
0x16	R_CAI	A(sym) Call absolute indirect (modifiable to call PC relative).
0x17	R_CREL	A(sym–*) Call PC relative modifiable to call absolute indirect.
0x08	R_BA	A(sym) Branch absolute (non-modifiable).
0x18	R_RBA	A(sym) Branch absolute (modifiable BA sym to BA fixed address).
0x19	R_RBAC	A(sym) Branch absolute constant (modifiable BA fixed address to BA sym).
0x0A	R_BR	A(sym–*) Branch relative to self (non-modifiable).
0x1A	R_RBR	A(sym–*) Branch relative to self (modifiable BR sym–* to BA fixed address).
0x1B	R_RBRC	A(sym) Branch absolute constant (modifiable BA fixed address to BR sym–*).
0x04	R_RTB	A((sym–*)/2) IAR relative branch (non–modifiable).
0x14	R_RRTBI	A((sym–*)/2) IAR relative branch (modifiable BALI to BA).
0x15	R_RRTBA	A(sym) Absolute branch (modifiable BA to BALI).

Implementation Specifics for the Relocation Information

Standard UNIX and AIX Version 2 practice is to retain relocation information only for unresolved references or references between distinct sections; once a reference is resolved (by link editing COFF files together), the relocation information is discarded. This is sufficient for an incremental bind and a fixed address space model. For the XCOFF file to provide rebind and a relocatable address space model, the following changes are incorporated:

- The relocation types are extended to provide the rebind capability.
- Resolved relocation items (RLDs) are not discarded.
- The relocation information for the .text and .data sections is used by the linkage editor and ignored by the loader.
- A special loader section is defined that contains any relocation data required by a dynamic relocating system loader.

Line Number Information for the a.out File

Line number entries are used by the symbolic debugger to debug code at the source level. When present, there is one line number entry for every source line that can have a breakpoint. The line numbers are grouped by function; the first entry for each function is l_lnno= 0, and the entry has a symbol table index of the function name in place of the physical address.

The **linenum.h** file, which defines the structure for line number entries, contains the following fields:

l_symndx	The symbol table index of the function name only if the value of the l_lnno field is 0.
l_paddr	The physical address of the line number if the value of the l_lnno field is not 0.
l_lnno	The line number relative to the start of the function(zero).

Special Data Sections for the a.out File

The following special sections are defined for an XCOFF file:

- Loader Section
- Debug Section
- Typchk Section
- Exception Section.

Loader Section

The loader data is produced by the linkage editor and contains the information required by the dynamic loader. All of the loader data is contained in the raw data portion of the .loader section.The loader section consists of the following parts:

- Header fields
- External symbol table
- Relocation table
- Import file IDs
- String table.

The **loader.h** file defines the structure for the .loader section.

Loader Header Fields

l_version	The .loader section version number. This value is currently 1.
l_nsyms	The number of symbol table entries in the .loader section.
l_nreloc	The number of relocation table entries in the .loader section.
l_istlen	The length of the .loader import file ID name string table.
l_nimpid	The number of import file IDs in the .loader import file ID name string table.
l_impoff	The offset from the beginning of the .loader section to the .loader import file ID name string table. Points to the length field of the first file ID.
l_stlen	The length of the .loader string table.
l_stoff	The offset from the beginning of the .loader section to the .loader string table. This field points to the length field of the first entry in the string table.

Loader External Symbol Table Fields

The **loader.h** file also contains external symbol table fields. Each entry in the .loader external symbol table is 24 bytes long. There are three implicit external symbols for the .text, .data,and .bss sections. These are referenced by way of symbol table index values of 0 through 2, respectively. The first symbol contained in the .loader external symbol table is referenced with an index value of 3. The external symbol table fields are as follows:

l_name	The null-padded symbol name if it is 8 characters or less in length.
l_zeros	A value of 0 indicates that the symbol name is in the .loader string table. This field overlays the l_name field.
l_offset	The offset from the beginning of the .loader string table to the first character of the symbol name (not the length field).This field overlays the second word of the l_name field.
l_value	The address value assigned by the binder to the symbol.
l_scnum	The number of the section that contains the symbol.If it is defined, the symbol will be in the .text, .data, or .bss sections. If the symbol is undefined, the section number field will be a value of 0.

l_smtype — The symbol type, import flag, export flag, and entry flag.

1	Imported symbol.
2	Entry point descriptor symbol.
3	Exported symbol.
4	Reserved.
5–7	Symbol type as defined by the XTY_*xx* **#define** statements in the **syms.h** header file.

l_smclas — The storage class of the symbol as defined by the XMC_*xx* **#define** statements in the **syms.h** header file.

l_ifile — The import file ID; the ordinal value of the import file name in the .loader import file ID name string table.

l_parm — The parameter type check field; the offset from the beginning of the .loader string table. This offset points to the first character of the parameter hash field (not to its length field).

Loader Relocation Table Fields

The **loader.h** file also contains the following relocation table fields. Each entry in the .loader relocation table is 12 bytes long. The structure for the first 10 bytes of each relocation entry is the same as the structure for a regular XCOFF relocation entry, which is defined in the **reloc.h** header file.

l_vaddr — The virtual address of the relocatable reference.

l_symndx — The loader symbol table index (nth entry) of the symbol that is being referenced. This field is zero-based where 0, 1, and 2 are implicit symbols for the the .text, .data, and .bss sections, respectively. Symbol index 3 is the index for the first symbol actually contained in the .loader symbol table.

l_rtype — The relocation type (The value of this field is same as the r_type field in the **reloc.h** header file.)

l_rsecnm — The number of the .text, .data, or .bss section being relocated (associated with l_vaddr field). This is a one-based index into the section header table.

Loader Import File ID Name String Fields

Each file ID string consists of a length field and a 3-part file name field consisting of the file path, base name, and archive member name. Each part of the file name field is null-delimited.The first import file ID is LIBPATH information to use in default search by the **exec** subroutine. The LIBPATH information consists of file paths separated by colons. There is no base name or archive member name, so the file path is followed by three null delimiters.

l_impidpath — The import file ID file path string, which is null-delimited.

l_impidbase — The import file ID file base name string, which is null-delimited.

l_impidmem — The import file ID archive file member name string,which is null-delimited.

Loader String Table Fields

The string table consists of the names of long symbols (greater than 8 characters in length) and the parameter type checking strings.

l_parmstlen The length of the string (parameter type-checking or long name) including the null, if present.

l_parmst The parameter type string (not null-delimited)or symbol name string (null-delimited).

Debug Section

The debug data is the dbx *stabstrings* (symbol table strings) and is produced by the compilers and assemblers. This data is referenced from the symbol table. A stabstring is a C-language character string that is null-terminated. Each string is preceded by a 2-byte field that indicates the length of the string to facilitate deletion. The .debug section contains only a raw data portion.

The following two fields are repeated for each dbx stabstring:

d_stabstlen The length of the string.

d_stabst The dbx stabstring.

Also refer to the discussion of dbx stabstring grammar.

Typchk Section

The type check data is the parameter type-checking hash strings that are produced by compilers and assemblers. The type check data is referenced from the symbol table. The .typchk section contains only a raw data portion.

The following two fields are repeated for each parameter type-checking string:

t_parmstlen The length of the string.

t_parmst The parameter type-checking hash string.

Type Encoding and Checking Format for Data

The type encoding and checking scheme provided facilitates detection of errors prior to execution of a program. Information about all external symbols (data and programs) is encoded and then checked for consistency at bind or load time. It is important to enforce the maximum checking required by the semantics of each particular language supported by AIX, yet still provide protection to applications written in more than one language.

The type encoding and checking mechanism features 4-part hash encoding that provides some flexibility in checking. The mechanism also incorporates a unique value, called UNIVERSAL, that matches any code. The UNIVERSAL hash can be used as an escape mechanism for assembly programs or for programs where type information or subroutine interfaces might not be known. The UNIVERSAL hash is 4 blank ASCII characters or a null pointer into the hash section.

The following 4 fields are associated with the type encoding and checking mechanism:

code length	A 2-byte field containing the length of the hash. This field has a value of 10.
language identifier	A 2-byte predefined code representing each language. These codes are the same as those defined for the TRACEBACK table defined by the e_lang field.
general hash	A 4-byte field representing the most general form by which a data symbol, or program, can be described. It is the greatest common denominator among languages supported by AIX. If incomplete or if no information is available, a universal hash should be generated. The general hash is language-independent, and must match in order for the binding to succeed.
language hash	A 4-byte field containing a more detailed, language-specific representation of what is in the general hash. It allows for the strictest type-checking desired by a given language. This part is used in intra-language binding and is not checked unless both symbols have the same language identifier.

Exception Section

The exception table contains trap instruction addresses, compiler language-identification codes, and trap reason codes produced by compilers and assemblers. The exception table is referenced from the symbol table. The .except section contains only a raw data portion.

The exception table has a similar organization as the line number table. Within the exception table there is one group of exception table entries per function.

An exception table entry with a reason code value of zero contains the symbol table index to the function (C_EXT or C_HIDEXT) symbol table entry. Reference from the symbol table to an entry in the exception table is via the function auxiliary symbol table entry. The x_tagndx field is a file pointer to the exception table for the function. This same auxiliary entry also has a file pointer to the line number table.

The **exceptab.h** file, which defines the structure for exception table entries, contains the following fields:

e_symndx	The symbol table index of the function name, if e_reason is zero.
e_paddr	The physical address of the trap instruction, if e_reason is not zero.

e_lang — An 8-bit field that contains the compiler language identifier.

0x00	C
0x01	Fortran
0x02	Pascal
0x03	Ada
0x04	PL/I
0x05	Basic
0x06	Lisp
0x07	Cobol
0x08	Modula2
0x09	C++
0x0A	RPG
0x0B	PL8, PLIX
0x0C	Assembly

e_reason — The compiler-dependent trap-exception reason code. Zero is not a valid trap-exception reason code.

Symbol Table for the a.out File

One composite symbol table is defined for the module. An XCOFF symbol table consists of a COFF symbol table as a base and includes the extensions for XCOFF. The symbol table contains information used by the linkage editor (external symbols) and by the symbolic debugger (function definitions and internal and external symbols).

For each external symbol, one or more auxiliary entries are defined that provide additional information about or for the external symbol.There are two major types of external symbols: those that represent replaceable units (csects) and those that represent externally invoked functions. For symbols that represent a replaceable unit (csect), an auxiliary entry contains the length and storage mapping class. For symbols that represent external invoked functions, an auxiliary entry contains references to: the csect in which they are contained, the parameter type checking information for the function, and the symbolic debugger information for the function.

The ordering of the symbol table must be arranged to accommodate both the symbolic debugger requirements and to permit effective management by the linkage editor of the different sections of the object file as a result of linkage edit actions such as garbage collection or rebinding.

The symbol table consists of at least one fixed length entry per symbol with some symbols followed by auxiliary entries of the same size.

There are .file symbol table entries used to bracket all the symbols of a source file. Any symbolic debugger information of file scope immediately follows the .file entry before the first csect entry. All symbolic debugger information between csect entries are associated with its containing csect. This ordering is required by the linkage editor so that if a csect is deleted or replaced, all symbolic debugger symbols associated with the csect can also be deleted or replaced.

All symbols, regardless of storage class and type, have the same primary format for their entries in the symbol table. Each entry in the symbol table is 18 bytes in length with the index for the first symbol table entry being zero.

The **syms.h** file, which defines the structure for the symbol table primary and auxiliary entries, contains the following fields:

n_name	The null-padded symbol name or symbolic debugger stab string if it is less than 8 characters in length. The n_sclass field is used to determine if the field is a symbol name or symbolic debugger stab string. By convention, an n_sclass value with the high order bit on indicates that this field is a symbolic debugger stab string.
n_zeroes	A value of zero indicates that the symbol name is in the string table or .debug section. This field overlays the n_name field.
n_offset	The offset into the string table or .debug section of the symbol name. The offset is relative to the start of the string table or .debug section. An offset value of zero indicates a null or zero-length symbol name. This field overlays the second word of the n_name field.
n_value	The symbol value. The symbol values are storage-class dependent.
n_scnum	The section number of the symbol.
n_type	The basic type and derived type. This is part of the original COFF definition used by the **sdb** symbolic debugger.
n_sclass	The storage class of the symbol. The **storclass.h** and **dbxstclass.h** header files contain the definitions for the storage classes.
n_numaux	The number of auxiliary entries for the symbol.

Symbol Table Storage Classes (n_sclass)

As defined for COFF plus the extensions for XCOFF, the n_sclass storage class field in the symbol table is defined to have one of the following values.

C_EFCN	–1	Physical end of form.
C_NULL	0	Null.
C_AUTO	1	Automatic variable.
C_EXT	2	External symbol.
C_STAT	3	Static.
C_REG	4	Register variable.
C_EXTDEF	5	External definition.
C_LABEL	6	Label.
C_ULABEL	7	Undefined label.
C_MOS	8	Member of a structure.
C_ARG	9	Function argument.
C_STRTAG	10	Structure tag.
C_MOU	11	Member of a union.
C_UNTAG	12	Union tag.
C_TPDEF	13	Type definition.
C_USTATIC	14	Uninitialized static.
C_ENTAG	15	Enumeration tag.
C_MOE	16	Member of enumeration.
C_REGPARM	17	Register parameter.
C_FIELD	18	Bit field.

C_BLOCK	100	Beginning or end of block.
C_FCN	101	Beginning or end of function.
C_EOS	102	End of structure.
C_FILE	103	File name.
C_LINE	104	Used only by utility programs.
C_ALIAS	105	Duplicate tag.
C_HIDDEN	106	Like static (used to avoid name conflicts).
C_HIDEXT	107	Unnamed external symbol (used to avoid name conflicts).
C_BINCL	108	Beginning of header file.
C_EINCL	109	End of header file.
C_GSYM	128	Global variable.
C_LSYM	129	Automatic variable allocated on stack.
C_PSYM	130	Argument to subroutine allocated on stack.
C_RSYM	131	Register variable.
C_RPSYM	132	Argument to function or procedure stored in the register.
C_STSYM	133	Statically allocated symbol.
C_BCOMM	135	Beginning of common block.
C_ECOML	136	Local member of common block.
C_ECOMM	137	End of common block.
C_DECL	140	Declaration of object.
C_ENTRY	141	Alternate entry.
C_FUN	142	Function or procedure.
C_BSTAT	143	Beginning of static block.
C_ESTAT	144	End of static block.

Storage Classes by Usage and Symbol Value Classification

Symbol classes that are used and relocated by the **ld** command and symbol values are addresses:

C_EXT	External symbol.
C_HIDEXT	Unnamed external symbol.

Symbol classes that are relocated by the **ld** command (symbol values are addresses):

C_BLOCK	Beginning or end of an inner block (.bb or .eb).
C_FCN	Beginning or end of a function (.bf or .ef only).
C_HIDDEN	Unnamed symbol.
C_LABEL	Label (contained in csect).
C_STAT	Static symbol (contained in statics csect).

Symbol classes that are used by the **ld** command and **dbx** for file scoping and accessing purposes:

C_FILE	The source file name. The n_value field holds the symbol index of the next file entry. The n_name field is the name of the file. For COFF compatibility, if there is an auxiliary entry, then the name of the file is in the auxiliary entry.
C_BINCL	The beginning of the header file. The n_value field is the line number offset in the object file to first line number from the header file.
C_EINCL	The end of the header file. The n_value field is the line number offset in the object file to last line number from the header file.

Symbol classes that exist only for **dbx** symbolic debugging purposes and are not relocated:

C_BCOMM	The beginning of the common block. The value of this field is not important, and the name is the name of the common block.
C_ECOML	The local member of the common block. The value is offset within the common block.
C_ECOMM	The end of the common block. The value of this field is not important.
C_BSTAT	The beginning of the static block. The value is the symbol table index of the csect that contains the static symbols. The name is .bs.
C_ESTAT	The end of the static block. The value of this field is not important. The name is .es.
C_DECL	The declaration of the object (type declarations). The value of this field is undefined.

Storage Classes by Usage and Symbol Value Classification (continued)

C_ENTRY	An alternate Fortran entry, which has a corresponding C_EXT symbol. The value of this field is undefined.
C_FUN	A function or procedure. This field might have a corresponding C_EXT symbol. The value of this field is offset within the csect in which it is contained.
C_GSYM	A global variable, which has a corresponding C_EXT symbol. The value of this field is undefined.
C_LSYM	An automatic variable allocated on the stack. The value of this field is offset relative to the stack frame (platform-dependent).
C_PSYM	An argument to a subroutine allocated on the stack. The value of this field is offset relative to the stack frame (platform-dependent).
C_RSYM	A register variable. The value of this field is the register number.
C_RPSYM	An argument to a function or procedure stored in a register. The value is this field is the number of the register where the argument is stored.
C_STSYM	A statically-allocated symbol. The value of this field is offset within the csect that is pointed to by the C_BSTAT entry in which it is contained.

Symbol classes that belong to COFF, but that are ignored by the **ld** command and **dbx**. These symbol classes are not relocated.

C_ARG	An argument to a function or procedure. This is replaced by C_PSYM.
C_AUTO	An automatic variable allocated on the stack. This is replaced by C_LSYM.
C_REG	A register variable. This is replaced by C_RSYM.
C_REGPARM	A register argument to a function or procedure. This is replaced by C_RPSYM.

COFF symbol storage classes that are superseded by the C_DECL storage class:

C_ENTAG	The enumeration tag.
C_EOS	The end of the structure.
C_FIELD	A field in the structure.
C_MOE	A member of the enumeration type.
C_MOS	A member of the structure.

C_MOU	A member of the union.
C_STRTAG	A structure tag.
C_TPDEF	A user-defined type declaration.
C_UNTAG	A union tag.

Also refer to the discussion of dbx stabstring grammar.

Symbol Values (n_value)

The definition of the n_value field of a symbol table is dependent on the storage class of the symbol.

C_AUTO	A stack offset.
C_EXT	A relocatable address.
C_STAT	A relocatable address.
C_REG	A register number.
C_LABEL	A relocatable address.
C_MOS	An offset.
C_ARG	A stack offset.
C_STRTAG	Zero.
C_MOU	An offset.
C_UNTAG	Zero.
C_TPDEF	Zero.
C_ENTAG	Zero.
C_MOE	An enumeration value.

Symbol Values (n_value) (continued)

C_REGPARM	A register number.
C_FIELD	A bit displacement.
C_BLOCK	A relocatable address.
C_FCN	A relocatable address.
C_EOS	A size.
C_FILE	A symbol table index to the next C_FILE (minus one for the last file). This is a one-way linked list in the symbol table.
C_ALIAS	A tag index.
C_HIDDEN	A relocatable address.
C_BINCL	An offset in the object file.
C_EINCL	An offset in the object file.
C_GSYM	Zero (undefined).
C_LSYM	An offset.
C_PSYM	An offset.
C_RSYM	A register number.
C_RPSYM	A register number.
C_STSYM	An offset.

C_BCOMM	Zero (undefined).
C_ECOML	An offset.
C_ECOMM	Zero (undefined).
C_DECL	Zero (undefined).
C_ENTRY	Zero (undefined).
C_FUN	An offset.
C_BSTAT	A symbol table index to the csect containing static symbols.
C_ESTAT	Zero (undefined).

The value of a relocatable symbol is equal to the virtual address of that symbol. When a section is relocated by the linkage editor, the values of these symbols change.

Section Numbers (n_scnum)

The definition of the n_scnum field of a symbol table is defined to have one of the following values:

N_DEBUG	–2	A special symbolic debugging symbol.
N_ABS	–1	An absolute symbol. The symbol is not relocatable.
N_UNDEF	0	An undefined or uninitialized external symbol. The symbol is undefined if the n_value field is zero. A relocatable external symbol that is not defined in the current file. The symbol is defined but uninitialized if the n_value field is non-zero. Then_value field provides the size of the symbol. The symbol is a relocatable external symbol.
N_SCNUM	X	All other values. The section number where the symbol was defined.

Section Numbers and Storage Classes

Symbols of certain storage classes are defined in the standard COFF to be restricted to certain section numbers. These are as follows:

C_AUTO	N_ABS.
C_EXT	N_ABS, N_UNDEF, N_SCNUM.
C_STAT	N_SCNUM.
C_REG	N_ABS.
C_LABEL	N_UNDEF, N_SCNUM.
C_MOS	N_ABS.
C_ARG	N_ABS.
C_STRTAG	N_DEBUG.
C_MOU	N_ABS.
C_UNTAG	N_DEBUG.
C_TPDEF	N_DEBUG.
C_ENTAG	N_DEBUG.
C_MOE	N_ABS.
C_REGPARM	N_ABS.
C_FIELD	N_ABS.

C_BLOCK	N_SCNUM.
C_FCN	N_SCNUM.
C_EOS	N_ABS.
C_FILE	N_DEBUG.
C_ALIAS	N_DEBUG.
C_HIDEXT	N_SCNUM.
C_BINCL	N_DEBUG.
C_EINCL	N_DEBUG.
C_GSYM	N_DEBUG.
C_LSYM	N_ABS.
C_PSYM	N_ABS.
C_RSYM	N_ABS.
C_RPSYM	N_ABS.
C_STSYM	N_ABS.
C_BCOMM	N_DEBUG.
C_ECOML	N_ABS.
C_ECOMM	N_DEBUG.
C_DECL	N_DEBUG.
C_ENTRY	N_DEBUG.
C_FUN	N_ABS.
C_BSTAT	N_DEBUG.
C_ESTAT	N_DEBUG.

Symbol Table Implementation Specifics

For AIX Version 3, the symbolic debugger to be supported is **dbx** (although other debuggers can be accommodated). Some items for symbolic debuggers that appeared in the COFF symbol table of previous versions are now in the .debug section and are referenced from the symbol table.

Also refer to the discussion of dbx stabstring grammar.

Symbol Table Auxiliary Entry Formats for the a.out File

The symbol table contains auxiliary entries to provide supplemental information for a symbol. The auxiliary entries for a symbol are adjacent to its symbol table entry. The size of each auxiliary entry is the same as a symbol table entry (18 bytes in length). The quantity and format of auxiliary entries depends on its type and storage class.

For XCOFF, the convention is that for external symbols that have more than one auxiliary entry, the last auxiliary entry will be the .csect auxiliary entry.

The following list summarizes the selection of the various auxiliary entry formats.

n_name	Storage Class	Auxiliary entry format
.file	C_FILE	File name
.text	C_STAT	Section
.data	C_STAT	Section
.bss	C_STAT	Section
symbol	C_EXT, C_HIDEXT	Function, csect

Auxiliary entries that relate to the symbolic debugger and all entries that are defined for the standard COFF are not included in this list.

Csect (External) Auxiliary Entry

The auxiliary entry to identify csects and entry points is one of the primary additions for the COFF extended definition. There is one csect (external) auxiliary entry for each external symbol. The convention is that the csect auxiliary entry will be the last auxiliary entry for a symbol that has more than one auxiliary entry.

x_scnlen — The meaning of this field is dependent on the x_symtype field, as follows:

If x_smtype is XTY_SD, then this is the csect length.

If x_smtype is XTY_LD, then this is the symbol table index of the containing csect.

If x_smtype is XTY_CM, then this is the csect length.

If x_smtype is XTY_ER, then this contains zero.

x_parmhash — An index to the parameter type-check hash in the .typchk section.

x_snhash — The .typchk section number.

x_smalgn — A 5-bit symbol (csect) alignment value (log 2).

x_smtype — A 3-bit symbol type field:

XTY_ER	0	External reference.
XTY_SD	1	Csect section definition.
XTY_LD	2	Entry point (label definition).
XTY_CM	3	Common (.bss).
XTY_EM	4	Error message (linkage editor use).
XTY_US	5	Reserved for internal use.

x_smclas — Storage mapping class used by the linkage editor for arranging csects.

XMC_PR	0	Read-only program code.
XMC_RO	1	Read-only constant.
XMC_DB	2	Read-only debug dictionary table.
XMC_GL	6	Read-only global linkage (Interfile interface code).
XMC_XO	7	Read-only extended operation (psuedo-machine instruction).
XMC_SV	8	Read-only supervisor call.
XMC_TI	12	Read-only traceback index csect.
XMC_TB	13	Read-only traceback table csect.
XMC_RW	5	Read or write data.
XMC_TCO	15	Read or write TOC anchor for (TOC addressability).
XMC_TC	3	Read or write general TOC entry.
XMC_DS	10	Read or write descriptor csect.
XMC_UA	4	Unclassified (treated as read or write).
XMC_BS	9	BSS class (uninitialized static internal).
XMC_UC	11	Unnamed Fortran common.

x_stab — Reserved.

x_snstab — Reserved.

Function Auxiliary Entry

The auxiliary table entry for a function is defined in standard COFF as follows:

x_tagndx — The tag index; a file pointer to the exception table.

x_fsize — The size (in bytes) of the function.

x_lnnoptr — A file pointer to the line number.

x_endndx — An index to the next entry beyond this function.

x_tvndx — Reserved.

File Name Auxiliary Entry

The auxiliary table entry for a file name is defined in standard COFF to contain a 14-character file name in bytes 0 through 13.The remaining bytes are zero. If a file name is longer than 14 characters, it is contained in the string table and the auxiliary entry contains an offset to the name in the string table. For XCOFF, the file name is placed in the name field of the C_FILE symbol table entry.

x_fname — The source file name.

x_zeroes — A value of zero indicates that the file name is in the string table. (This field overlays the x_fname field.)

x_offset — An offset from the beginning of the string table to the first character of the file name. (This field overlays the second word of x_fname.)

dbx Stabstring Grammar (C, COBOL, Pascal, FORTRAN, and Modula–2)

In the following grammar, there are 5 types of terminal symbols denoted by all CAPS. These types are NAME, STRING, INTEGER, HEXINTEGER, and REAL. These are described by the regular expressions below:

NAME	Any set of characters excluding ; : ' "
STRING	'*' or "*" where \" and \' can be used inside the string
INTEGER	(–)[**0–9**]+
HEXINTEGER	[**0–9A–F**]+
REAL:	[**+–**][**0–9**]+(.)[**0–9**]*([**eE**](**+–**)[**0–9**]+) | (**+–**)**INF** | **QNAN** | **SNAN**

Notation: [ ] implies one instance, [ ]* implies zero or more instances, [ ]+ implies one or more instances, and ( ) implies zero or one instances.

What follows is a the grammar for a stabstring. Except in the case of a constant whose value is a string, there are no blanks in a stabstring. This is very much like the 4.3 BSD stabstring grammar, with the notable exception being that stabstrings that exceed 8 characters in length are placed in a .debug section rather than in the string section.

Stabstring: Basic structure of stabstring

NAME : *Class*	Name of object followed by object classification
: *Class*	Unnamed object classification

Class: Object classifications

c = *Constant* ;	Constant object
NamedType	User-defined types and tags
Parameter	Argument to subprogram
Procedure	Subprogram declaration
Variable	Variable in program
Label	Label object

Constant: Constant declarations

b *OrdValue*	Boolean constant
c *OrdValue*	Character constant
e *TypeId* , *OrdValue*	Enumeration constant
i INTEGER	Integer constant
r REAL	Floating point constant
s STRING	String constant
S *TypeId* , *NumElements* , *NumBits* , *BitPattern*	Set constant

OrdValue: Associated numeric value

INTEGER

dbx Stabstring Grammar (continued)

NumElements: Number of elements in the set

INTEGER

NumBits: Number of bits in item

INTEGER

NumBytes: Number of bytes in item

INTEGER

BitPattern: Hexadecimal representation, up to 32 bytes

HEXINTEGER

NamedType: User-defined types and tags

t *TypeId*	User–defined type (TYPE or typedef)
T *TypeId*	Struct, union, or enumeration tag

Parameter: Argument to procedure or function

a *TypeId*	Passed by reference in general register
p *TypeId*	Passed by value on stack
v *TypeId*	Passed by reference on stack
C *TypeId*	Constant passed by value on stack
D *TypeId*	Passed by value in floating point register
R *TypeId*	Passed by value in general register

Procedure: Procedure or function declaration

Proc	Procedure at current scoping level
Proc , NAME , NAME	Procedure named 1st NAME, local to 2nd NAME, where 2nd NAME is different than the current scope

Variable: Variable in program

TypeId	Local (automatic) variable of type *TypeId*
d *TypeId*	Floating register variable of type *TypeId*
r *TypeId*	Register variable of type *TypeId*
G *TypeId*	Global (external) variable of type *TypeId*
S *TypeId*	Module variable of type *TypeId* (C static global)
V *TypeId*	Own variable of type *TypeId* (C static local)

Label: Label

L	Label name

dbx Stabstring Grammar (continued)

Proc: Different types of functions and procedures

f *TypeId*	Private function of type TypeId
m *TypeId*	Module (Modula–2, ext. Pascal)
J *TypeId*	Internal function of type *TypeId*
F *TypeId*	External function of type *TypeId*
I	Internal procedure
P	External procedure
Q	Private procedure

TypeId: Type declarations and identifiers

INTEGER — Type number of previously-defined type

INTEGER = *TypeDef*
New type number described by *TypeDef*

INTEGER = *TypeAttrs TypeDef*
New type with special type attributes

Note: Type attributes (*TypeAttrs)* are extra information associated with a type, such as alignment constraints or pointer checking semantics. dbx recognizes only the size attribute and the packed attribute. The size attribute denotes the total size of a padded element within an array. The packed attribute indicates that a type is a packed type. Any other attributes are ignored by dbx.

TypeAttrs: Any additional information, ignored by dbx

@ TypeAttrList ;

TypeAttrList: List of special type attributes

TypeAttrList ; *@ TypeAttr*
TypeAttr

TypeAttr: Special type attributes

a INTEGER	Align boundary
s INTEGER	Size in bits
p INTEGER	Pointer class (for checking)
P	Packed type
Other	Anything not covered is skipped entirely

dbx Stabstring Grammar (continued)

TypeDef: Basic descriptions of objects

INTEGER	The type number of a previously defined type
b *TypeId* ; *NumBytes*	Pascal space type
c *TypeId* ; *NumBits*	Complex type *TypeId*
d *TypeId*	File of type *TypeId*
e *EnumList* ;	Enumerated type (default size is 32 bits)
g *TypeId* ; *NumBits*	Floating-point type of size *NumBits*

For **i** types, *ModuleName* refers to the Modula-2 module from which it is imported.

i NAME : NAME ;	Imported type ModuleName:Name
i NAME : NAME , *TypeId* ;	Imported type ModuleName:Name of type *TypeId*
n *TypeId* ; *NumBytes*	String type with max string length *Bytesize*
o NAME ;	Opaque type
o NAME , *TypeId*	Opaque type with definition of *TypeId*
w	Wide character
z *TypeId* ; *NumBytes*	Pascal gstring type
C *Usage*	COBOL Picture
K *CobolFileDesc*	COBOL File Descriptor
M *TypeId* ; *Bound*	Multiple instance type of *TypeId* with length *Bound* (character*3 = M–2;3)
N	Pascal *Stringptr*
S *TypeId*	Set of type *TypeId*
* *TypeId*	Pointer of type *TypeId*
Array	
Subrange	
ProcedureType	For function *types* rather than *declarations*
Record	Record, structure, union, or group types

EnumList: List of enumerated scalars

Enum
EnumList Enum

Enum: Enumerated scalar description

NAME : *OrdValue* ,

dbx Stabstring Grammar (continued)

Array: Array descriptions

a *TypeDef* ; *TypeId*
Array

A *TypeId* Open array of *TypeId*

D INTEGER ; *TypeId*
N-dimensional dynamic array of *TypeId*

E INTEGER ; *TypeId*
N-dimensional subarray of *TypeId*

P *TypeDef* ; *TypeId*
Packed Array

Subrange: Subrange descriptions

r *TypeId* ; *Bound* ; *Bound*
Subrange type (char, int,...), lower and upper bounds

Bound: Upper and Lower Bound descriptions

INTEGER Constant bound

Boundtype INTEGER
Variable or dynamic bound, value is address of or offset to bound

J Bound is indeterminable (no bounds)

Boundtype: Adjustable subrange descriptions

A Bound passed by reference on stack

T Bound passed by value on stack

a Bound passed by reference in register

t Bound passed by value in register

ProcedureType: Function variables (1st type C only; others Modula-2 & Pascal)

f *TypeId* ; Function returning type *TypeId*

f *TypeId* , *NumParams* ; *TParamList* ;
Function of N parameters returning type *TypeId*

p *NumParams* ; *TParamList* ;
Procedure of N parameters

R *NumParams* ; *NamedTParamList* ;
Pascal subroutine parameter

F *TypeId* , *NumParams* ; *NamedTParamList* ;
Pascal function parameter

NumParams:
Number of parameters in routine

INTEGER

dbx Stabstring Grammar (continued)

TParamList: Types of parms in Modula-2 func variable

TParam Type of parameter and passing method
TParamList TParam

TParam: Type and passing method

TypeId , *PassBy* ;

NamedTParamList:
Types of parms in Pascal routine variable

NamedTParam
Type of parameter and passing method
NamedTParamList NamedTParam

NamedTParam:
Named type and passing method

Name : *TypeId* , *PassBy* ;

Record: Types of structure declarations

s *NumBytes FieldList* ;
Structure or record definition
u *NumBytes FieldList* ;
Union
v *NumBytes FieldList VariantPart* ;
Variant Record
G *Redefinition* , **n** *NumBits FieldList* ;
COBOL Group without conditionals
Gn *NumBits FieldList* ;
G *Redefinition* , **c** *NumBits Condition FieldList* ;
COBOL Group with conditionals
Gc *NumBits Condition FieldList* ;

FieldList: Structure content descriptions

Field Member of record or union
FieldList Field

Field: Structure member type description

NAME : *TypeId* , *BitOffset* , *NumBits* ;

VariantPart: Variant portion of variant record

[*Vtag VFieldList*]
Variant description

dbx Stabstring Grammar (continued)

VTag: Variant record tag

(*Field* Member of variant record
(NAME : ; Variant key name

VFieldList: Variant record content descriptions

VList Member of variant record
VFieldList VList

VList: Variant record fields

VField Member of variant record
VField VariantPart

VField: Variant record member type description

(*VRangeList* :
Variant with no field list
(*VRangeList* : *FieldList*
Variant with field list

VRangeList: List of variant field labels

VRange Member of variant record
VRangeList , *VRange*

VRange: Variant field descriptions

b *OrdValue* Boolean variant
c *OrdValue* Character variant
e *TypeId* , *OrdValue*
Enumeration variant
i INTEGER Integer variant
r *TypeId* ; *Bound* ; *Bound*
Subrange variant

BitOffset: Offset in bits from beginning of structure

INTEGER

Usage: Cobol usage description

PICStorageType NumBits , *EditDescription* , *PicSize* ;
Redefinition , *PICStorageType NumBits* , *EditDescription* , *PicSize* ;
PICStorageType NumBits , *EditDescription* , *PicSize* , *Condition* ;
Redefinition , *PICStorageType NumBits* , *EditDescription* , *PicSize* ,
Condition ;

dbx Stabstring Grammar (continued)

Redefinition: Cobol redefinition

r NAME

PICStorageType:
Cobol PICTURE types

a	Alphabetic
b	Alphabetic edited
c	Alphanumeric
d	Alphanumeric edited
e	Numeric signed trailing included
f	Numeric signed trailing separate
g	Numeric signed leading included
h	Numeric signed leading separate
i	Numeric signed default comp
j	Numeric unsigned default comp
k	Numeric packed decimal signed
l	Numeric packed decimal unsigned
m	Numeric unsigned comp-x
n	Numeric unsigned comp-5
o	Numeric signed comp-5
p	Numeric edited
q	Numeric unsigned
s	Indexed item
t	Pointer

EditDescription: Cobol edit description

STRING	Edit characters in an alpha PIC
INTEGER	Decimal point position in a numeric PIC

PicSize: Cobol description length

INTEGER	Number of repeated '9's in numeric clause or length of edit format for edited numeric

Condition: Conditional variable descriptions

NAME : INTEGER = **q** *ConditionType* , *Valuelist* ;

ConditionType: Condition descriptions

ConditionPrimitive , *KanjiChar*

dbx Stabstring Grammar (continued)

ConditionPrimitive:
Primitive type of Condition

n *Sign DecimalSite*
Numeric conditional

a	Alphanumeric conditional
f	Figurative conditional

Sign: For types which have explicit sign

+	Positive
–	Negative
0	No explicit sign value

DecimalSite: Number of places from left for implied decimal point

INTEGER

KanjiChar: 0 only if Kanji character in value

INTEGER

ValueList Values associated with condition names

Value
ValueList Value

Value Values associated with condition names

INTEGER : STRING
Integer indicates length of string

CobolFileDesc: COBOL file description

Organization AccessMethod NumBytes

Organization: COBOL file description organization

i	Indexed
l	Line Sequential
r	Relative
s	Sequential

AccessMethod: COBOL file description access method

d	Dynamic
o	Sort
r	Random
s	Sequential

dbx Stabstring Grammar (continued)

PassBy: Parameter passing method

INTEGER 0 = passed-by reference; 1 = passed-by value

Export: Modula-2 export description

INTEGER *ExportInfo*
Identifier and type description

ExportInfo: Modula-2 only

c = *Constant* Constant

f *TypeId* , *NumParams* ; *ParamList* ;
Function

p *NumParams* ; *ParamList* ;
Procedure

t *TypeId*

v *TypeId*

ParamList: Func/Proc parameter declarations

Param

ParamList Param

Param: Parameter name, type, and passing method

NAME : *TypeId* , *PassBy* ;

Note: It might become necessary to split a stabstring among multiple symbol table entries for ease of handling long stabstrings. A ? or \ indicates that the symbol information is continued in the next stab entry. This continuation can occur after the separator (usually ; , for a *EnumList*) in *List* entries (*ParamList, EnumList, FieldList,* ...). Also, continuation is allowed after the ; in *Array* and *Subrange* grammar entries.

String Table for the a.out File

The string table contains the names of symbols that are longer than 8 characters. If present, the first 4 bytes contain the length (in bytes) of the string table, including the length bytes. Thus, offsets into the string table are greater than or equal to 4. The remainder of the table is a sequence of null-terminated ASCII strings. If the n_zeroes field in a symbol table entry is zero, then_offset field gives the offset into the string table of the name for the symbol.

File

/usr/include/a.out.h The path to the **a.out.h** header file.

Related Information

The **as** command, **dbx** command, **dump** command, **ld** command,**nm**command, **size** command, **strip** command, and **what** command.

ar File Format

Purpose

Describes the AIX indexed-archive file format.

Description

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

Fixed-Length Header

Each archive begins with a fixed-length header that contains offsets to special archive file members. The fixed-length header also contains the magic number, which identifies the archive file. The fixed-length header has the following format:

```
#define AIAMAG "<aiaff>\n" /* Magic string */
#define SAIAMAG 8          /* Length of magic string */

struct fl_hdr              /* Fixed-length header */
{
   char fl_magic[SAIAMAG]; /* Archive magic string */
   char fl_memoff[12];     /* Offset to member table */
   char fl_gstoff[12];     /* Offset to global symbol table */
   char fl_fstmoff[12];    /* Offset to first archive member */
   char fl_lstmoff[12];    /* Offset to last archive member */
   char fl_freeoff[12];    /* Offset to first mem on free list */

};
```

The AIX indexed-archive file format uses a double-linked list within the archive file to order the file members; therefore, file members may not be sequentially ordered within the archive. The offsets contained in the fixed-length header are used to locate the first and last file members of the archive. Member order is determined by the linked list.

The fixed-length header also contains the offsets to the member table, global symbol table, and the free list. Both the member table and the global symbol table exist as members of the archive and are kept at the end of the archive file. The free list contains file members that have been deleted from the archive. When adding new file members to the archive, this area is used before the archive file size is expanded. A zero offset in the fixed-length header indicates that the member is not present in the archive file.

File Member Header

Each archive file member is preceded by a file member header, which contains the following information about the file member:

```
#define AIAFMAG "'\n"                   /* Header trailer string */
   struct ar_hdr                        /* File member header */
   {
           char ar_size[12];            /* File member size - decimal */
           char ar_nxtmem[12];          /* Next member offset - decimal*/
           char ar_prvmem[12];          /* Previous member offset -dec */
           char ar_date[12];            /* File member date - decimal */
           char ar_uid[12];             /* File member user id-decimal */
           char ar_gid[12];             /* File member group id - dec */
           char ar_mode[12];            /* File member mode - octal */
           char ar_namlen[4];           /* File member name length - dec
   */
           union
              {
              char ar_name[2];          /* Start of member name */
              char ar_fmag[2];          /* AIAFMAG - string to end */
              };
           _ar_name;                    /* Header and member name */};
```

The member header provides support for member names up to 255 characters long. The ar_namlen field contains the length of the member name. The character string containing the member name begins at the_ar_name field. The AIAFMAG string is cosmetic only.

Each archive member header begins on an even-byte boundary.The total length of a member header is (sizeof (struct ar_hdr) plus ar_namlen). The actual data for a file member begins at the first even-byte boundary beyond the member header and continues for the number of bytes specified by the ar_size field. The **ar** command inserts null bytes for padding where necessary.

All information in the fixed-length header and archive members is in printable ASCII. Numeric information, with the exception of the ar_mode field, is stored as decimal numbers; the ar_mode field is stored in octal. Thus, if the archive file contains only printable files, you can print the archive.

Member Table

A member table is always present in an indexed archive file. This table is used to quickly locate members of the archive. The fl_memoff field in the fixed-length header contains the offset to the member table. The member table member has a zero length name. The **ar** command automatically creates and updates (but does not list) the member table. A member table contains the following information:

- The number of members. This is 12 bytes long and stored in ASCII as a decimal number.
- The array of offsets into the archive file. The length is 12 times the number of members. Each offset is 12 bytes long and stored in ASCII as a decimal number.
- The name string table. The size is (ar_size minus (12 times (the number of members plus 1))). That is, the total length of the member minus the length of the offsets minus the length of the number of members.

The string table contains the same number of strings as there are offsets. All strings are null-terminated. Each offset from the array corresponds sequentially to a name in the string table.

Global Symbol Table

If an archive file contains XCOFF object-file members that are not stripped, the archive file will contain a global symbol table member. This global symbol table is used to locate file members that define global symbols. The **strip** command can be used to delete the global symbol table from the archive. The fl_gstoff field in the fixed-length header contains the offset to the global symbol table. The global symbol table member has a zero length name. The **ar** command automatically creates and updates (but does not list) the global symbol table. A global symbol table contains the following information:

- The number of symbols. This is 4 bytes long and can be accessed with the **sgetl** command and the **sputl** command.
- The array of offsets into the archive file. The length is 4 times the number of symbols. Each offset is 4 bytes long and can be accessed with the **sgetl** command and the **sputl** command.
- The name string table. The size is (ar_size minus (4 times (the number of symbols plus 1))), that is, the total length of the member minus the length of the offsets minus the length of the number of symbols.

The string table contains the same number of strings as there are offsets. All strings are null-terminated. Each offset from the array corresponds sequentially to a name in the string table.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/ar.h	The path to the **ar.h** header file.

Related Information

The **sgetl**, **sputl** subroutine.

The **a.out** file format.

The **ar** command, **ld** command, **strip** command.

audit File Format

Purpose

Describes the auditing data structures.

Description

The **audit.h** header file contains definitions for the auditing system, including commands and structures for the following subroutines and daemons:

audit
auditbin
auditevents
auditlog
auditobj
auditproc

The **audit.h** header file defines the arguments for these subroutines. This header file also defines the structure of a bin as generated by the kernel.

Audit Bin Format

The format of the audit bin is described by the **aud_bin** structure. An audit trail consists of a sequence of bins, each of which must start with a bin head and end with a bin tail. The **aud_bin** structure contains the following fields:

bin_magic — The magic number for the bin (0xf0f0).

bin_version — The version number for the bin (0).

bin_tail — Indicates whether the bin describes the audit trail head or tail:

- 0 Identifies the bin header.
- 1 Identifies the bin end (tail).
- 2 Identifies the trail end.

bin_len — The (unpacked) length of the bin's records. A non-zero value indicates that the bin has a tail record.

bin_plen — The current length of the bin's record (might be packed).

bin_time — The time at which the head or tail was written.

bin_reserved1 — Not currently used.

bin_reserved2 — Not currently used.

Audit Record Format

Each audit record consists of a list of fixed-length event identifiers, each of which can be followed by a variable-length tail. The format of the audit record is described by the **aud_rec** structure, which contains the following fields to identify the event:

ah_event[16] — The name of the event and a null terminator.

ah_length — The length of the tail portion of the audit record.

ah_result — An indication of whether the event describes a successful operation. The values for this field are:

0	Indicates sucessful completion.
1	Indicates a failure.
>1	An errno describing the failure.

ah_prepend — Indicates whether or not this is a prepend record.

The **aud_rec** structure also contains the following fields to identify the user and the process:

ah_ruid — The real user id; that is, the id number of the user who created the process that wrote this record.

ah_luid — The login id of the user who created the process that wrote this record.

ah_name[16] — The program name of the process, along with a null terminator.

ah_pid — The process id of the process that wrote this record.

ah_ppid — The process id of the parent of this process.

ah_time — The time at which this audit record was written.

The record tail follows this header information.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

File

/usr/include/sys/audit.h — The path to the **audit.h** header file.

Related Information

The **audit** subroutine, **auditbin** subroutine, **auditevents** subroutine, **auditlog** subroutine, **auditobj** subroutine, **auditproc** subroutine, **auditwrite** subroutine.

The **audit** command, **auditcat** command, **auditbin** command, **auditpr** command, **auditselect** command, **auditstream** command.

core File Format

Purpose

Contains an image of a process at the time of an error.

Description

A **core** file is created in the current directory when various errors occur. Errors such as memory-address violations, illegal instructions, bus errors, and user-generated quit signals commonly cause this *core dump.* The **core** file that is created contains a memory image of the terminated process. A process with an effective user ID that is different from the real user ID does not produce a memory image. The contents of a core dump are organized as follows:

core_dump

c_signo
c_flag
c_entries
*c_tab
c_stack
c_size

c_u (the actual u_block)

ld_info

user mode stack

user data (optional)

The **core_dump** structure, defined by the **core.h** header file, occurs at the beginning of a **core** file. The **core_dump** structure includes the following fields:

c_signo — The number of the error signal, which indicates the error that caused the core dump.

c_flag — One of the following flags, which describe the core dump type:

FULL_CORE	The core contains the data sections (0x01).
UBLOCK_VALID	The u_block has been dumped (0x10).
USTACK_VALID	The user stack has been dumped (0x20).
LE_VALID	The core contains at least one module (0x40).
CORE_TRUNC	The core was truncated (0x80).

c_entries — The number of core dump modules.

*c_tab — A pointer to the **ld_info** structure, which defines the core table (loader modules).

c_stack — A pointer to the user mode stack.

c_size — The size of the user stack.

The c_u field follows this information in the core dump. The c_u field contains the *user structure* (a copy of the actual u_block), which includes the registers as they existed at the time of the fault.

The **ld_info** structure and then the user mode stack follow the u_block in the core dump.

By default, the user data is not included in a core dump. This partial core dump includes the current process's stack and user structure and the state of the registers at the time of the fault. A partial core dump contains sufficient information for a stack traceback. The size of a core dump can also be limited by the **setrlimit** subroutine.

To enable a full core dump, set the SA_FULLDUMP flag in the **sigaction** subroutine for the signal that is to generate a full core dump. If this flag is set when the core is dumped, the data section is included in the core dump.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Files

/usr/include/sys/core.h	The path to the **core.h** header file.
/usr/include/sys/param.h	The path to the **param.h** header file, which describes AIX operating system parameters used by the hardware.
/usr/include/sys/reg.h	The path to the **reg.h** header file, which defines user registers used by the AIX operating system.
/usr/include/sys/user.h	The path to the **user.h** header file, which describes the user structure and contains process information that is not needed unless a process is running.

Related Information

The **raise** subroutine, **setgid** subroutine, **setrlimit** subroutine, **setuid** subroutine, **sigaction** subroutine.

The **adb** command, **dbx** command.

The **param.h** header file.

cpio File Format

Purpose

Describes the copy in/out (**cpio**) archive file.

Description

The **cpio** utility is used to back up and recover files. The files are saved on the backup medium in the **cpio** format.

When the **cpio** command is used with the **–c** flag, the header for the **cpio** structure can be read as follows:

```
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.

When the **–c** flag of the **cpio** command is not used, the header structure contains the following fields:

h_magic — Contains the constant octal 070707 (or 0x71c7).

h_dev — The device that contains a directory entry for this file.

h_ino — The i-node number that identifies the input file to the file system.

h_mode — The mode of the input file, as defined in the **mode.h** header file.

h_uid — The user ID of the owner of the input file.

h_gid — The group ID of the owner of the input file.

For remote files, these fields contain the ID after reverse translation.

h_nlink — The number of links that are connected to the input file.

h_rdev — The ID of the remote device from which the input file is taken.

h_mtime — The time when data was last modified. For remote files, this field contains the time at the server. This time is changed by the **creat**, **fclearf**, **truncate**, **mknod**, **openx**, **pipe**, **utime**, and **writex** subroutines.

h_namesize — The length of the path name, including the terminating null byte.

h_filesize — The length of the file in bytes. This is the length of the data section that follows the header structure.

h_name — The null-terminated path name. The length of the path name, including the null byte, is indicated by the *n variable*, where *n* equals ((h_namesize % 2) + h_namesize). In other words, the *n* variable is equal to the h_namesize field if the h_namesize field is even. If the h_namesize field is odd, the *n* variable is equal to the h_namesize field + 1.

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

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Files

/usr/include/sys/mode.h	The path to the **mode.h** header file, which defines the interpretation of a file mode.
/usr/include/sys/stat.h	The path to the **stat.h** header file, which defines the data structure returned by the status subroutines.

Related Information

The **creat** subroutine, **fclear** subroutine, **ftruncate** subroutine, **mknod** subroutine, **openx** subroutine, **pipe** subroutine, **scanf** subroutine, **utime** subroutine, **writex** subroutine.

The **mode.h** file, **stat.h** file.

The **cpio** command, **find** command.

nterm File Format

Purpose

Terminal driving tables for the **nroff** command.

Description

The **nroff** command uses driving tables to customize its output for various types of output devices, such as printing terminals, special word–processing terminals (such as Diablo, Qume, or NEC Spinwriter mechanisms), or special output filter programs. These driving tables are written as ASCII files, and are installed in the **/usr/lib/nterm/tab.***Name* file, where the *Name* parameter is the name for a terminal type.

The first line of a driving table should contain the name of the terminal: simply a string with no imbedded white space (any combination of spaces, tabs, and newlines). The next part of the driver table is structured as follows:

- **bset** [*Integer*]
- **breset** [*Integer*]
- **hor** [*Integer*]
- **vert** [*Integer*]
- **newline** [*Integer*]
- **char** [*Integer*]
- **em** [*Integer*]
- **halfline** [*Integer*]
- **adj** [*Integer*]
- **twinit** [*Character String*]
- **twrest** [*Character String*]
- **twnl** [*Character String*]
- **hlr** [*Character String*]
- **hlf** [*Character String*]
- **flr** [*Character String*]
- **bdon** [*Character String*]
- **bdoff** [*Character String*]
- **iton** [*Character String*]
- **itoff** [*Character String*]
- **ploton** [*Character String*]
- **plotoff** [*Character String*]
- **up** [*Character String*]
- **down** [*Character String*]
- **right** [*Character String*]
- **left** [*Character String*]

The meanings of these fields are as follows:

bset Bits to set in the **c_oflag** field of the termio structure before output.

breset Bits to reset in the **c_oflag** field of the termio structure before output.

hor Horizontal resolution in units of 1/240 of an inch.

vert Vertical resolution in units of 1/240 of an inch.

newline	Space moved by a newline (linefeed) character in units of 1/240 of an inch.
char	Quantum of character sizes, in units of 1/240 of an inch (that is, a character is a multiple of **char** units wide).
em	Size of an em in units of 1/240 of an inch.
halfline	Space moved by a half–linefeed (or half–reverse–linefeed) character in units of 1/240 of an inch.
adj	Quantum of white space, in 1/240 of an inch, (that is, white spaces are a multiple of **adj** units wide). Note: If this is less than the size of the space character, the **nroff** command outputs fractional spaces using plot mode. Also, if the **–e** switch to the **nroff** command is used, the **adj** variable is set equal to the **hor** variable by the **nroff** command.
twinit	Sequence of characters used to initialize the terminal in a mode suitable for the **nroff** command.
twrest	Sequence of characters used to restore the terminal to normal mode.
twnl	Sequence of characters used to move down one line.
hlr	Sequence of characters used to move up one–half line.
hlf	Sequence of characters used to move down one–half line.
flr	Sequence of characters used to move up one line.
bdon	Sequence of characters used to turn on hardware boldface mode, if any.
bdoff	Sequence of characters used to turn off hardware boldface mode, if any.
iton	Sequence of characters used to turn on hardware italics mode, if any.
itoff	Sequence of characters used to turn off hardware italics mode, if any.
ploton	Sequence of characters used to turn on hardware plot mode (for Diablo–type mechanisms), if any.
plotoff	Sequence of characters used to turn off hardware plot mode (for Diablo–type mechanisms), if any.
up	Sequence of characters used to move up one resolution unit (**vert**) in plot mode, if any.
down	Sequence of characters used to move down one resolution unit (**vert**) in plot mode, if any.
right	Sequence of characters used to move right one resolution unit (**hor**) in plot mode, if any.
left	Sequence of characters used to move left one resolution unit (**hor**) in plot mode, if any.

This part of the driving table is fixed–format, and you cannot change the order of entries. You should put entries on separate lines, and these lines should contain exactly two fields (no comments allowed) separated by white space. For example,

```
bset    0
breset  0
Hor     24
```

Follow this first part of the driving table with a line containing the word *charset*, and then specify a table of special characters that you want to include. That is, specify all the non–ASCII characters that the **nroff** command knows by two–character names, such as `\(hy`. If the **nroff** command does not find the word *charset* where it expects to, it terminates processing with an error message.

Each definition in the part after *charset* occupies one line, and has the following format:

```
chname width output
```

where `chname` is the (two–letter) name of the special character, `width` is its width in ems, and `output` is the string of characters and escape sequences to send to the terminal to produce the special character.

For **nls** fonts, *chname* can also be an NLesc sequence, for instance, `\<c,>`.

International Character Support:

For NLS printers using NLS fonts, the characters XX is required in the *charset*. XX provides the width of single–width NLS characters, respectively, in the "width" columns. The "output" columns for this character must contain a single question mark, ?.

Japanese Language Support:

For SJIS printers using **kanji** fonts, the characters X1 and X2 are required in the *charset*. X1 and X2 provide the width of single–width and double–width SJIS characters, respectively, in their "width" columns. The "output" columns for these characters must contain a single question mark, ?.

If any field in the *charset* part of the driving table does not pertain to the output device, you can give that particular sequence as a null string or leave out the entry. Special characters that do not have a definition in this file are ignored on output by the **nroff** command.

You can put the *charset* definitions in any order, so it is possible to speed up the **nroff** command by putting the most used characters first. For example,

```
charset
em 1–
hy 1–
\–1–
bu 1 +\bo
```

The best way to create a terminal table for a new device is to take an existing terminal table and edit it to suit your needs. Once you create such a file, put it in the **/usr/lib/nterm** directory, and give it the name **tab.***xyz* where *xyz* is the name of the terminal and the name that you pass the **nroff** command by way of the **–T** flag (for example, **nroff –Txyz**).

Implementation Specifics

This command is part of Formatting Tools in the Text Formatting System of AIX for RISC System/6000.

File

/usr/lib/nterm/tab.*Name*	Terminal files.

Related Information

The **nroff** command.

profile File Format

Purpose

Sets the user environment at login time.

Description

The **profile** file contains commands to be executed at login and variable profile 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 profile 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 the **mkuser** command 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
```

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/profile	Profile file.
$HOME/.profile	Individual user profile file.

Related Information

The **bsh** command, **csh** command, **env** command, **login** command, **mail** command, **mkuser** command, **ksh** command, **stty** command, and **su** command.

sccsfile File Format

Purpose

Describes the format of a Source Code Control System (SCCS) file.

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.

Several lines in an SCCS file 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 that does not begin with the control character contains text from the source file and cannot begin with the control character.

The following paragraphs describe each logical part of an SCCS file.

Checksum

The checksum is the first line of an SCCS file. This line has the following format:

```
@hNumber
```

The control character and variables in the checksum line have the following meanings:

@h	Designates a magic number of 064001 octal (or 0x6801).
Number	Represents the sum of all characters in the SCCS file (not including the characters in this line).

Delta Table

The delta table consists of a variable number of entries such as:

```
@sNumber/Number/Number
@d Type SID Date Time UID Number Number
@i Number . . .
@x Number . . .
@g Number . . .
@m Number
@c Comments . . .
@e
```

The control characters and variables in the delta table entries have the following meanings:

@s Designates the first line of each entry, which contains the number of lines inserted, deleted, and unchanged (respectively).

@d Designates the second line of each entry, which contains:

- *Type* The type of delta. D designates a normal delta; R designates a removed delta.
- *SID* The SCCS ID (SID) of the delta.
- *Date* The date, in the *yy/mm/dd* format, at which the delta was created.
- *Time* The time, in the *hh:mm:ss* format, at which the delta was created.
- *UID* The login name that corresponds to the real user ID at the time the delta was created.
- *Number* The serial numbers of the delta and its predecessor, respectively.

@i Indicates the serial numbers of the deltas that are included in this file. This line may contain several delta numbers. This line is optional.

@x Indicates the serial numbers of the deltas that are excluded from this file. This line can contain several delta numbers. This line is optional.

@g Indicates the serial numbers of the deltas that are ignored. This line can contain several delta numbers. This line is optional.

@m Indicates a modification request (MR) number associated with the delta. There can be several MR lines in an SCCS file, each one containing a different MR number. These lines are optional.

@c Comment lines associated with the delta. There can be several comment lines in an SCCS file. These lines are optional.

@e Ends the delta table entry.

User Names

The list of login names and numerical group IDs of users who can add deltas to the file. The names and IDs are separated by new-line characters. This section uses the following control characters:

@u A bracketing line that indicates the beginning of a user name list. This line appears before the first line in the list.

@U A bracketing line that indicates the end of a user name list. This line appears after the last line in the list.

An empty list allows any user to make a delta.

Flags

Flags are keywords that are used internally in the SCCS system. The format of each line is:

`@fFlag Text`

The control character and variables in the checksum line have the following meanings:

@f Flag — Designates one of the following defined flags:

- **@ft** — Type of program. Defines the replacement for the `%Y%` identification keyword.
- **@fv** — Program name. Controls prompting for MR numbers in addition to comments upon delta creation. If a value is assigned, it defines an MR number validity-checking program.
- **@fi** — ID keywords. Controls the `No ID keywords` error warning message. When this flag is not set, the message is only a warning. When this flag is set, the absence of ID keywords will cause an error and the delta will fail.
- **@fb** — Branch. Allows the use of the **–b** option of the **get** command to cause a branch in the delta tree.
- **@fm** — Module name. Defines the replacement module name for the `%M%` identification keyword. This value is used to override the default.
- **@ff** — Floor. Defines the lowest release number from 0 through 9999 that can be retrieved by a **get** command for editing. This release number is called the *floor* release number.
- **@fc** — Ceiling. Defines the highest release number from 0 through 9999 that can be retrieved by a **get** command for editing. This release number is called the *ceiling* release number.
- **@fd** — Default SCCS ID. Defines the default SID to be used when one is not specified with a **get** command. When this flag is not set, the **get** command uses the most recently created delta.
- **@fn** — No changes. Causes the **delta** command to insert null deltas (delta entries with no changes) for any 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, skipped releases will be omitted from the delta table.
- **@fj** — Joint edit. Causes the **get** command to allow concurrent edits of the same base SID.
- **@fl** — Lock releases. Defines a list of releases that cannot be edited with **get** using the **–e** flag.
- **@fq** — User defined flag. Defines the replacement for the `%Q%` identification keyword.

Text — Indicates optional text.

Comments

When rthe comments are taken from a file containing descriptive text using the **admin** command with the **–t** option, the contents of that file are displayed in this section. Typically, the comments section contains a description of the purpose of the delta. This section uses the following control characters:

@t A bracketing line that indicates the beginning of the comments section. This line appears before the first comment line.

@T A bracketing line that indicates the end of the comments section. This line appears after the last comment line.

Body

The body section consists of control and text lines. Control lines begin with the control character; text lines do not. This section contains the following types of control lines:

@I*Number* Indicates an insert control line. The serial number that corresponds to the delta for the control line is indicated by the *Number* parameter.

@D*Number* Indicates a delete control line. The serial number that corresponds to the delta for the control line is indicated by the *Number* parameter.

@E*Number* Indicates an end control line. The serial number that corresponds to the delta for the control line is indicated by the *Number* parameter.

File

/usr/bin/sccs The path to the **sccs** command, which is the administration program for the SCCS commands.

Related Information

The **admin** command, **delta** command, **get** command, **prs** command.

troff File Format

Purpose

Describes the output language from the **troff** command.

Description

The device-independent **troff** file format outputs a pure ASCII description of a typeset document. The description specifies the typesetting device, the fonts, and the point sizes of characters to be used as well as the position of each character on the page. A list of all the legal commands follows. Most numbers are denoted by the *Number* parameter and are ASCII strings. Strings inside of **[ ]** (brackets) are optional. The **troff** command can produce them, but they are not required for the specification of the language. The **\n** character has the standard meaning of newline character. Between commands, white space has no meaning. Whitespace characters are spaces and new lines.

The following are the legal commands:

s*Number* — Specifies the point size of the characters to be generated.

f*Number* — The font mounted in the specified position is to be used. The *Number* parameter value ranges from 0 to the highest font presently mounted. 0 is a special position, called by the **troff** command, but not directly accessible to the **troff** command user. Normally fonts are mounted starting at position 1.

c*Character* — Generates the specified character at the current location on the page; the value specified by the *Character* parameter is a single ASCII character.

C*XYZ* — Generates the *XYZ* special character. The name of the character is delimited by white space. The name is one of the special characters legal for the typesetting device as specified by the device specification found in the *DESC* file. This file resides in a directory specific for the typesetting device. (See the **troff font** file format and the **/usr/lib/font/dev*** directory.)

H*Number* — Changes the horizontal position on the page to the number specified. The number is in basic units of motions as specified by the *DESC* file. This is an absolute **goto** statement.

h*Number* — Adds the number specified to the current horizontal position. This is a relative **goto** statement.

V*Number* — Changes the vertical position on the page to the number specified (down is positive).

v*Number* — Adds the number specified to the current vertical position.

NumberCharacter — This is a two-digit number followed by an ASCII character. The meaning is a combination of the **h***Number* command followed by the **c***Character* command. The specified number is added to the current horizontal position and then the ASCII character, specified by the *Character* parameter, is produced. This is the most common form of character specification.

n*B A* — Indicates that the end of a line has been reached. No action is required, though by convention the horizontal position is set to 0. The **troff** command specifies a resetting of the *x,y* coordinates on the page before printing more characters. The first number, *B*, is the amount of space before the line and the second number, *A*, the amount of space after the line. The second number is delimited by white space.

w — A **w** command appears between words of the input document. No action is required. It is included so that one device can be emulated more easily on another device.

p*Number* — Begins a new page. The new page number is included in this command. The vertical position on the page should be set to 0.

#...\n — A line beginning with a # (pound sign) is a comment.

Dl *X Y* — Draws a line from the current location to *X,Y*.

Dc *D***\n** — Draws a circle of the diameter specified by the *D* parameter with the leftmost edge being at the current location (*X,Y*). The current location after drawing the circle is *X+D,Y*, the rightmost edge of the circle.

De*DX DY***\n** — Draws an ellipse with the specified axes. The *DX* parameter is the axis in the *X* direction and the *DY* parameter is the axis in the *Y* direction. The leftmost edge of the ellipse is at the current location. After drawing the ellipse the current location is *X+DX,Y*.

Da *DH1 DV1 DH2 DV2***\n** — Draws a counterclockwise arc from the current position to the *DH1l+DH2, DV1+DV2* parameter that has a center of *DH1, DV1* from the current position. The current location after drawing the arc is at its end.

D~ *X Y X Y* ...**\n** — Draws a spline curve (wiggly line) between each of the *X,Y* coordinate pairs starting at the current location. The final location is the final *X,Y* pair of the list.

x i[nit]\n — Initializes the typesetting device. The actions required are dependent on the device. An **init** command always occurs before any output generation is attempted.

x T *Device***\n** — The name of the typesetter is specified by the *Device* parameter. This is the same as the parameter to the **–T** flag. The information about the typesetter is found in the **/usr/lib/font/dev***Device* directory .

x r[es] *N H V***\n** — The resolution of the typesetting device in increments per inch is specified by the *N* parameter. Motion in the horizontal direction can take place in units of basic increments specified by the *H* parameter. Motion in the vertical direction can take place in units of basic increments specified by the *V* parameter.

x p[ause]\n — Pause. Causes the current page to finish but does not relinquish the typesetter.

x s[top]\n Stop. Causes the current page to finish and then relinquishes the typesetter. Performs any shutdown and bookkeeping procedures required.

x t[railer]\n Generates a trailer. On some devices no operation is performed.

x f[ont] *N Font*\n Loads the specified font into position *N*.

x H[eight] *N*\n Sets the character height to *N* points. This causes the letters to be elongated or shortened. It does not affect the width of a letter. Not all typesetters can do this.

x S[lant] *N*\n Sets the slant to *N* degrees. Only some typesetters can do this and not all angles are supported.

The following commands are effective on the international extended characters:

Q*C1C2* Outputs the character specified by the two bytes specified by the *C1* and *C2* parameters. The high-order bits can be set in these bytes.

q*C* Outputs the character *C* parameter ORed with 0x80.

K*Digit1Digit2Digit3Digit4*
Outputs the character specified by the four hex digits *Digit1, Digit2, Digit3,* and *Digit4*. (*Digit1* is the high digit of the first byte; *Digit4* is the low digit of the second byte.)

k*Digit1Digit2* Outputs the byte specified by the two hex digits *Digit1* and *Digit2*. (*Digit1* is the high byte.)

Japanese Language Support: The preceding commands listed for international extended characters are also effective on SJIS extended characters.

Files

/usr/lib/font/dev*Name*/*DESC*.**out**	Description file for phototypesetter specified by *Name*.
/usr/lib/font/dev*Name*/*Font*.**out**	Font description files for phototypesetter specified by *Name*.

Related Information

The **troff** command.

The **troff Font** file format.

troff Font File Format

Purpose

Description files for the **troff** command.

Description

For each phototypesetter that the **troff** command supports and that is available on your system, there is a directory that contains files describing the phototypesetter and its fonts. This directory is named **/usr/lib/font/dev***Name*, where *Name* is the name of the phototypesetter. For a list of supported devices, see "Terminal Names for Phototypesetter and Comparable Devices".

For a particular phototypesetter *Name*, the ASCII *DESC* file in the **/usr/lib/font/dev***Name* directory within the **troff** command source directory describes its characteristics. A binary version of this file is found in the **/usr/lib/font/dev***Name*/*DESC*.**out** file. Each line of this ASCII file starts with a word that identifies a characteristic, followed by appropriate specifiers. Blank lines and lines beginning with the # character are ignored.

The legal lines for the *DESC* file are:

res *Number* Resolution of device in basic increments per inch.

unitwidth *Number*
Point size in which all width tables in the font description files are given. The **troff** command automatically scales the widths from the **unitwidth** size to the point size with which it is working.

sizescale *Number*
Scaling for fractional point sizes. *Number* is 1. The **sizescale** line is not currently used.

paperwidth *Number*
Width of paper in basic increments.

paperlength *Number*
Length of paper in basic increments.

biggestfont *Number*
The maximum number of characters in a font.

sizes *Number1 Number2...*
List of point sizes available on typesetter, ended by 0.

fonts *Number Name...*
Number of initial fonts, followed by the ASCII names of the fonts. For example:

```
fonts 4 R I B S
```

charset This is last keyword in the file and is on a line by itself. Following it is the list of special character names for this device. Names are separated by a space or a newline. The list can be as long as necessary. Names not in this list are not allowed in the font description files.

hor *Number* Smallest unit of horizontal motion.

vert *Number* Smallest unit of vertical motion.

The **hor** and **vert** lines describe the relationships between motions in the horizontal and vertical directions. For example, if the device moves in single basic increments in both directions, both the **hor** and **vert** lines have values of 1. If vertical motion only occurs in multiples of two basic units and horizontal motion only in one, **vert** is 2 and **hor** is 1.

For each font supported by the phototypesetter, there is also an ASCII file with the same name as the font (for instance, **R**, **I**, **CW**) that describes it. The format for a font description file is:

name *Name* Name of the font, such as **R** or **CW**.

internalname *Name*
Internal name of the font.

special Sets the flag indicating that the font is special.

ligatures *Name...*0
Sets the flag indicating that the font has ligatures. The list of ligatures follows and is ended by a zero. Accepted ligatures are **ff fi fl ffi ffl**.

spacewidth *Number*
Specifies width of space if something other than default (1/3 of an em) is desired.

nls The font contains international extended characters. See the **charset** line for a description of the character set in an **nls** font.

kanji The font contains SJIS extended characters. See **Japanese Language Support** for a description of the character set in a **kanji** font.

charset The character set must come at the end. Each line following the **charset** word describes one character in the font. Each line has one of two formats:

```
Name Width Kerning Code
Name "
```

where *Name* is either a single ASCII character or a special character name from the list found in the *DESC* file. For an **nls** font, *Name* can also be an NLesc sequence, for instance, `\<c,>`. The *Width* is in basic increments. The *Kerning* field is **1** if the character descends below the line, **2** if it rises above the letter `'a'`, and **3** if it both rises and descends. The *Code* field is the number sent to the typesetter to produce the character. For an **nls** font, *Code* can be a multi-byte sequence.

For fonts for extended-character output devices, *Code* can be a multi-byte sequence that begins and ends with a double quotation mark. In the sequence, control or non-printing characters can be represented by the following escape sequences: \n for newline; \r for return; \t for tab; \b for backspace; \" for double quote; \xdd for a hexadecimal number, where dd is two hexadecimal digits; and \ooo for an octal number, where ooo is three octal digits.

The second format, (*Name "*), is used to show that the character has more than one name. The double quotation marks shows that this name has the same values as the preceding line. The *Kerning* and *Code* fields are not used if the *Width* field is a double quotation mark character. The total number of different characters in this list should not be greater than the value of the **biggestfont** line in the *DESC* file.

Implementation Specifics

Japanese Language Support: For fonts for Japanese-language output devices, *Code* is a multi-byte sequence, as previously described.

For a **kanji** font, the only valid characters are the following:

X1	Represents all single-width SJIS characters
X2	Represents all double-width SJIS characters

All single-width SJIS characters are the same size as the **X1** character, and all double-width SJIS characters are the same size as the **X2** character. The *Code* field is not provided for **kanji** fonts; SJIS code is assumed to be the normal code of the printer. If the output printer does not use the SJIS code set, the postprocessor translates the character codes.

Files

/usr/lib/font/dev*Name*/*DESC*.**out**	Description file for phototypesetter specified by *Name*.
/usr/lib/font/dev*Name*/*Font*.**out**	Font description files for phototypesetter specified by *Name*.

Related Information

The **troff** command.

The **troff** file format.

utmp, wtmp, failedlogin File Format

Purpose

Describes formats for user and accounting information.

Description

The **utmp** file, the **wtmp** file, and the **failedlogin** file contain records with user and accounting information.

When a user logs in successfully, the **login** program writes entries in two files:

- The **/etc/utmp** file, which contains a record of users logged into the system.
- The **/usr/adm/wtmp** file (if it exists), which contains connect–time accounting records.

On an invalid login attempt, due to an incorrect login name or password, the **login** program makes an entry in:

- The **/etc/security/failedlogin** file, which contains a record of unsuccessful login attempts.

When you login as the **root** user, if there are any entries in the **/etc/security/failedlogin** file, you see a message advising you to check the file.

The records in these files follow the **utmp** format, defined in the **utmp.h** header file. An example of the **utmp** file format follows:

Example

```
#define UTMP_FILE        "/etc/utmp"
#define WTMP_FILE      "/usr/adm/wtmp"
#define FAILEDLOGIN_FILE "/etc/security/failedlogin"

#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 */

   stnet termio ut_termio;    /* Save login termio parameters */
   };
```

```
        /* 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 */
#define TSH_PROCESS  10
#define UTMAXTYPE TSH_PROCESS /* 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 TIME_MSG       "new time"
```

Implementation Specifics

This file format is part of Accounting Services in AIX BOS Extensions 2.

Files

/etc/utmp	Contains a record of users logged into the system.
/usr/adm/wtmp	Contains connect accounting information.
/etc/security/failedlogin	Contains a record of invalid login attempts.

Related Information

The **init** command, **login** command, **su** command.

The **utmp.h** file.

To see the steps you must take to establish an Accounting System, refer to How to Set Up an Accounting System in *General Concepts and Procedures*.

For more information about the Accounting System, the preparation of daily and monthly reports, and the accounting files, refer to Accounting Overview in *General Concepts and Procedures*.

ATE ate.def File Format

Purpose

Determines default settings for use in asynchronous connections and file transfers.

Description

The **ate.def** file sets the defaults for use in asynchronous connections and file transfers. It is created in the current directory the first time a user runs the Asynchronous Terminal Emulation (ATE) program. The **ate.def** file contains the default values the ATE program uses for the following:

- Data transmission characteristics
- Local system features
- Dialing directory file
- Control keys.

The first time the ATE program runs from a particular directory, it creates the **ate.def** file in that directory, with settings as follows:

```
LENGTH          8
STOP            1
PARITY          0
RATE            1200
DEVICE          tty0
INITIAL ATDT
FINAL
WAIT            0
ATTEMPTS        0
TRANSFER        p
CHARACTER       0
NAME            kapture
LINEFEEDS       0
ECHO            0
VT100           0
WRITE           0
XON/XOFF        1
DIRECTORY       /usr/lib/dir
CAPTURE_KEY     002
MAINMENU_KEY    026
PREVIOUS_KEY    022
```

The system user can edit the **ate.def** file with any ASCII text editor to permanently change the values of these characteristics. The values of these characteristics can also be *temporarily* changed with the ATE **alter** and **modify** subcommands, which can be accessed from either ATE Main Menu.

When you start the ATE program, it looks for a file named **ate.def** in the current directory. If the program finds the **ate.def** file, it changes the system's defaults to the values in the file.

Note: Type parameter names in uppercase letters. Spell them exactly as they appear in the original default file.

Define only one parameter per line.

If you define an incorrect value for a parameter in the **ate.def** file, you receive a system message while running the ATE program. However, the program continues to run using the default value.

Parameters in the ate.def File

LENGTH — Specifies the number of bits in a data character. This length must match the length expected by the remote system.

Options: 7 or 8.

Default: 8.

STOP — Specifies the number of stop bits appended to a character to signal that character's end during data transmission. This number must match the number of stop bits used by the remote system.

Options: 1 or 2.

Default: 1.

PARITY — Checks whether a character is successfully transmitted to or from a remote system. Must match the parity of the remote system.

For example, if you select even parity, when the number of 1 bits in the character is odd, the parity bit is turned on to make an even number of 1 bits.

Options: 0=none, 1=odd, or 2=even.

Default: 0.

RATE — Determines the number of bits transmitted per second (baud rate). The speed must match the speed of your modem and that of the remote system.

Options: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19200.

Default: 1200.

DEVICE — Specifies the name of the asynchronous port used to make a connection to a remote system.

Options: Locally created port names.

Default: tty0.

INITIAL — Defines the dial prefix, a string that must precede the telephone number when you autodial with a modem. For the proper dial commands, consult the user's guide for your modem.

Options: ATDT, ATDP, or other values, depending on the type of modem used.

Default: ATDT.

FINAL — Defines the dial suffix, a string that must follow the telephone number when you autodial with a modem. For the proper dial commands, consult the user's guide for your modem.

Options: Blank (none) or a valid modem suffix.

Default: no default.

WAIT — Specifies the time to wait between redialing attempts. The wait period does not begin until the connection attempt times out or until you interrupt it. If the ATTEMPTS parameter is set to 0 (zero), no redial attempt occurs.

Options: 0 (none) or a positive integer designating the number of seconds to wait.

Default: 0.

ATTEMPTS — Specifies the maximum number of times the ATE program tries to redial to make a connection. If the ATTEMPTS parameter is set to 0 (zero), no redial attempt occurs.

Options: 0 (none) or a positive integer designating the number of attempts.

Default: 0.

TRANSFER — Defines the type of asynchronous protocol that transfers files during a connection.

p **pacing**:
A file transfer protocol that controls the data transmission rate by waiting for a specified character or for a certain number of seconds between line transmissions. This helps prevent loss of data when the transmission blocks are either too large or sent too quickly for the system to process.

x **xmodem**:
An 8-bit file transfer protocol that detects data transmission errors and retransmits the data.

Options: p for **pacing**, x for **xmodem**.

Default: p.

CHARACTER — Specifies the type of **pacing** protocol to be used.

Character — Signal to transmit a line. Select one character.

When the **send** subcommand encounters a linefeed character while transmitting data, the subcommand waits to receive the pacing character before sending the next line.

When the **receive** subcommand is ready to receive data, it sends the pacing character, then waits 30 seconds to receive data. The **receive** subcommand sends a pacing character again whenever it finds a carriage return character in the data. The **receive** subcommand ends when it receives no data for 30 seconds.

Interval Number of seconds the system waits between each line it transmits. The value of the *Interval* variable must be an integer. The default value is 0, indicating a pacing delay of 0 seconds.

Default: 0.

NAME File name for incoming data (capture file).

Options: A valid AIX file name that is less than 40 characters long.

Default: `kapture`

LINEFEEDS Adds a line feed character after every carriage return character in the incoming data stream.

Options: 1 (on) or 0 (off).

Default: 0.

ECHO Displays the user's typed input.

For a remote computer that supports echoing, each character you send returns and is displayed on your screen. When the ECHO parameter is on, each character is displayed twice: first when it is entered, then when it returns over a connection. When the ECHO parameter is off, each character is displayed only once: when it returns over the connection.

Options: 1 (on) or 0 (off).

Default: 0.

VT100 The local console emulates a DEC VT100 terminal so DEC VT100 codes can be used with the remote system. With the VT100 parameter off, the local console functions like a workstation.

Options: 1 (on) or 0 (off).

Default: 0.

WRITE Captures incoming data and routes it to the file specified in the NAME parameter as well as to the display. Carriage return or line feed combinations are converted to line feed characters before they are written to the capture file. In an existing file, data is appended to the end of the file.

Note: The CAPTURE_KEY key sequence can be used to toggle capture mode on or off during a connection.

Options: 1 (on) or 0 (off).

Default: 0.

XON/XOFF Controls data transmission at a port as follows:

- When an Xoff signal is received, transmission stops.
- When an Xon signal is received, transmission resumes.

- An Xoff signal is sent when the receive buffer is nearly full.
- An Xon signal is sent when the buffer is no longer full.

Options: 1 (On), or 0 (Off).

Default: 1.

DIRECTORY Names the file that contains the user's dialing directory.

Default: the **/usr/lib/dir** file.

CAPTURE_KEY
Defines the control key sequence that toggles capture mode. When pressed, the CAPTURE_KEY key sequence starts or stops capturing (saving) the data that is displayed on the screen during an active connection.

Options: Any ASCII control character.

Default: ASCII octal 002 (STX). This is the Ctrl-B key sequence on the workstation keyboard.

MAINMENU_KEY
Defines the control key sequence that returns the Connected Main Menu so you can issue a command during an active connection. The MAINMENU_KEY control key sequence functions only from the connected state.

Options: Any ASCII control character.

Default: ASCII octal 026 (SYN). This is the Ctrl-V key sequence on the workstation keyboard.

PREVIOUS_KEY
Defines the control key sequence that displays the previous screen anytime during the program. The screen that is displayed varies, depending on the screen in use when you press the PREVIOUS_KEY key sequence.

Options: Any ASCII control character.

Default: ASCII octal 022 (DC2). The ASCII control character is mapped to the AIX interrupt signal. This is the Ctrl-R key sequence on the workstation keyboard.

Note: Changing or remapping the control keys may be necessary if you have two applications in which control keys conflict. For example, if the control keys mapped for the ATE program conflict with those in your text editor, you may want to remap the ATE control keys.

The ASCII control character you select may be in octal, decimal, or hexadecimal format, as follows:

octal	000 through 037. The leading zero is required.
decimal	0 through 31.
hexadecimal	0x00 through 0x1F. The leading 0x is required. The x may be uppercase or lowercase.

Example

To change characteristics of ATE emulation, create an **ate.def** file that defines those characteristics.

For example, to change the RATE to `300 bps`, the DEVICE to `tty3`, the TRANSFER mode to `x` (**xmodem** protocol), and the DIRECTORY to `my.dir`, create the following **ate.def** file in the directory from which you run the ATE program:

```
RATE          300
DEVICE        tty3
TRANSFER      x
DIRECTORY     my.dir
```

Next time you start the ATE program from that directory, it uses the values you have defined in place of the defaults.

Implementation Specifics

This file is part of Asynchronous Terminal Emulation in BOS Extensions 2.

File

/usr/lib/dir Contains the default dialing directory file.

Related Information

How to Edit the ATE Default File in in *Communication Concepts and Procedures* explains the steps to follow to change the **ate.def** file. How to Set Up an ATE Dialing Directory in in *Communication Concepts and Procedures* explains how to create a dialing directory file.

The **ate** command starts ATE and creates the **ate.def** file. The subcommands of **ate** include the **alter** subcommand, **connect** subcommand, **directory** subcommand, **modify** subcommand, **send** subcommand, and **receive** subcommand.

ATE Overview in *Communication Concepts and Procedures* introduces the ATE program.

ATE Overview for System Management in *Communication Concepts and Procedures* discusses tasks involved in managing ATE and lists the aspects of ATE that can be customized.

Using Control Keys with ATE in *Communication Concepts and Procedures* explains how the control keys are used.

Using the ATE Unconnected Main Menu and Using the ATE Connected Main Menu in *Communication Concepts and Procedures* explain how the menus work and which commands are accessible from each menu.

ATE Dialing Directory File Format

Purpose

Lists phone numbers that the ATE program uses to establish modem connections.

Description

The ATE dialing directory file lists phone numbers that the Asynchronous Terminal Emulation (ATE) system uses to establish remote connections over modems.

The dialing directory file can be named with any valid AIX name and established in any AIX directory to which the user has read and write access. The dialing directory file can be edited with any ASCII text editor. The default dialing directory is the **/usr/lib/dir** file.

The **connect** and **directory** subcommands of ATE access the dialing directory file. Use the **directory** subcommand to view the dialing directory.

You can have more than one dialing directory. To change the directory file the ATE program will use, modify the **ate.def** file in your current directory. (You can also enter phone numbers with the **connect** command if they are not in your dialing directory.)

Note: The dialing directory file can contain up to 20 lines (one entry per line); ATE ignores subsequent lines.

Format of Dialing Directory File entries

The dialing directory is similar to a page in a telephone book. It contains entries for the remote systems you call with the ATE program. The format of a dialing directory entry is:

Name Phone Rate Length StopBit Parity Echo Linefeed

The fields must be separated by at least one space. More spaces can be used to make the entry easier to read. The fields are:

Name — Name that identifies a telephone number. The name can be any combination of 20 or fewer characters. Use the _ (underscore) instead of a blank between words in a name; for example, `data_bank`.

Phone — The telephone number to be dialed. The number can be 40 characters or fewer. Consult the user's guide for your modem for a list of acceptable digits and characters. For example, if you must dial a 9 to access an outside line, include a `9` and a `,` (comma) before the telephone number as follows: `9,1112222`.

Note: Although the telephone number can be up to 40 characters long, the **directory** subcommand displays only the first 26 characters.

Rate — Transmission or baud rate (bits per second). Determines the number of characters transmitted per second. Select a baud rate that is compatible with the communication line you are using. The following are acceptable rates: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19200.

Length — Number of bits that make up a character. The entry for the *Length* field can be 7 or 8.

StopBit — Stop bits that signal the end of a character. The entry for the *StopBit* field can be 1 or 2.

Parity Checks whether a character was successfully transmitted to or from a remote system. The entry for the *Parity* field can be 0 (none), 1 (odd), or 2 (even).

Echo Determines whether typed characters display locally. The entry for the *Echo* field can be 0 (echo off) or 1 (echo on).

Linefeed Adds a linefeed character at the end of each line of data coming in from a remote system. The linefeed character is similar in function to the carriage return and newline characters. The entry for the *Linefeed* field can be 0 (linefeed off) or 1 (linefeed on).

Example

Following is a sample dialing directory entry:

```
CompuAid        111-0000   1200   7   1   2   0   0
```

In this example, `CompuAid` is the *Name*, `111-0000` is the *Phone*, `1200` is the *Rate*, `7` is the *Length*, `1` is the *StopBit*, `2` is the *Parity*, the first `0` (zero) is the *Echo*, and the second `0` (zero) is the *Linefeed*.

Implementation Specifics

This file is part of Asynchronous Terminal Emulation in BOS Extensions 2.

Files

ate.def Contains ATE default values.
/usr/lib/dir Contains the default dialing directory listing.

Related Information

How to Set Up a Dialing Directory File for ATE in *Communication Concepts and Procedures* explains the steps to follow in creating a customized dialing directory file and instructing ATE to use it instead of the default listing.

How to Edit the ATE Default File in *Communication Concepts and Procedures* explains how to set up the **ate.def** file for your defaults.

The **ate** command starts the ATE program. The ATE **connect** subcommand and **directory** subcommand read the dialing directory file.

ATE Overview in *Communication Concepts and Procedures* introduces the ATE program.

ATE Overview for System Management in *Communication Concepts and Procedures* discusses tasks involved in managing ATE and lists the aspects of ATE that can be customized.

BNU Devices File Format

Purpose

Contains information about devices on the local system that can establish a connection to a remote computer using the Basic Networking Utilities (BNU) program.

Description

The **/usr/lib/uucp/Devices** file contains information about the devices on the local system that can establish a connection to a remote computer using the Basic Networking Utilities (BNU) program. This file includes information for hardwired, telephone, and TCP/IP communication links.

Note: Only someone with root user authority can edit the **Devices** file, which is owned by the **uucp** login ID.

Fields in the Devices File

The **Devices** file must contain a description of each device on the local system that can establish a remote connection using the BNU program. Each line in the **Devices** file includes the following fields:

Type Typically specifies the type of hardwired or automatic calling unit (ACU) device.

Line Specifies the device name for the port.

Line2 Specifies the dialer name if the *Line* entry specifies an 801 dialer.

Class Typically specifies the transmission speed.

Dialer-Token Pairs
Specifies a particular type of autodialer (modem) and the token (a defined string of characters) that is passed to the dialer. Valid entries for this field are defined in the **/usr/lib/uucp/Dialers** file.

The fields appear on the line as follows:

Type Line Line2 Class Dialer-Token Pairs

There must be an entry in every field of a line in the **Devices** file. If a field does not apply to the particular type of device, system, use a – (hyphen) as a placeholder.

Lines in the **Devices** file cannot wrap. Each entry must be on only one line in the file. However, the **Devices** file can contain blank lines and comment lines. Comment lines begin with a # (pound sign). Blank lines are ignored.

Type **Field**

Enter one of the following keywords in this field:

Keyword	Explanation
ACU	Use this keyword, entered in uppercase letters, if your site connects multiple systems over the telephone network using automatic calling units (autodialers or modems).
Direct	Use this keyword, beginning with an uppercase D, if your site uses hardwired lines to connect multiple systems.

TCP Use this keyword, in uppercase letters, if your site uses TCP/IP.

SystemName Enter the name of a particular remote system hardwired to the local system. The *SystemName* keyword is the name assigned to each individual system, such as `hera`, `zeus`, or `merlin`.

This field corresponds to the *Type* field in the **/usr/lib/uucp/Systems** file.

Line Field

Type the device name for the line, or port, used in the communication link. For example, use the appropriate device name for a hardwired line, such as `tty1`. For a line connected to an ACU (a modem), use a device name appropriate to the dialer, such as `tty1` or `tty2`. For a TCP connection, enter a hyphen as a placeholder.

Line2 Field

Unless you are using an 801 dialer, type a – (hyphen) in this field as a placeholder. If you are using an 801 dialer, put the device name of the 801 ACU in this field. For example, if the entry in the *Type* field is `ACU` and the *Line* field entry (specifying the modem) is `tty1`, the *Line2* field entry (specifying the 801 dialer for the modem) might be `tty3` or `tty4`.

Note: The *Line2* field is used only to support older modems that require 801-type dialers. The modem is plugged into one serial port, and the 801 dialer is plugged into a separate serial port.

Class Field

For an ACU or a hardwired line, the *Class* field can be the speed of the device. In this case, for a hardwired line, type the transmission rate of the device connecting the two systems. For a telephone connection, type the speed at which the ACU transmits data, such as `300` or `1200` bps.

This field can also contain a letter with a speed (for example, `C1200`, `D1200`) to differentiate between classes of dialers. For example, some offices have more than one telephone network, one for internal use and one for external communications. In such a case, it is necessary to distinguish which lines should be used for each connection.

The *Class* field in the **Devices** file is matched against the *Class* field in the **/usr/lib/uucp/Systems** file. For example, if the **Systems** file entry for system `hera` is:

```
hera Any ACU 1200 3–3–5–2 ogin: nuucp ssword: oldoaktree
```

BNU searches for an entry in the **Devices** file with a *Type* of `ACU` and a *Class* of `1200`.

Some devices can be used at several specific speeds. In this case, make multiple entries for the device, specifying each speed on a separate line in the **Devices** file. If BNU cannot connect at the first speed, it will try the following speeds successively.

If a device can be used at any speed, type the word `Any` in the *Class* field. Note that the `A` in `Any` must be uppercase.

For a TCP/IP connection, enter a – (hyphen) as a placeholder.

Dialer-Token Pair Field

The *Dialer-Token Pair* field specifies a particular type of autodialer (modem) and the token (a defined string of characters) that is passed to the dialer. Valid entries for this field are defined in the **/usr/lib/uucp/Dialers** file.

For a hardwired connection, enter the word `direct` (note the lowercase `d`) as the *Dialer* entry and leave the *Token* entry blank.

For a telephone connection, enter the type of dialer and the token that is passed on to that modem. The *Token* field entry is either a telephone number or a predefined string used to reach the dialer.

For a telephone connection, enter one of the following as the *Dialer* field entry:

Entry	Definition
`hayes`	A Hayes dialer.
Other Dialers	Other dialers that you can specify by including the relevant information in the **/usr/lib/uucp/Dialers** file.
`TCP`	A TCP/IP connection. Enter `TCP` in the *Dialer* field entry if you have also entered `TCP` in the *Type* field.

Each *Dialer* field entry included as part of a *Dialer-Token Pair* field in the **Devices** file has a corresponding entry in the **Dialers** file.

If the *Token* field entry represents a telephone number, enter one of the following in the *Token* field to specify how the BNU program should use the telephone number listed in the **/usr/lib/uucp/Systems** file:

Entry	Definition
`\D`	The default token in a *Dialer-Token Pair* field. The `\D` token specifies that the BNU program should take the phone number listed in the **/usr/lib/uucp/Systems** file and pass it to the appropriate *dialer script* (entry) in the **/usr/lib/uucp/Dialers** file *without* including a dial-code abbreviation.
`\T`	This token instructs the BNU program to process the phone number by including the data specified in the **/usr/lib/uucp/Dialcodes** file. **Note:** If you are using dial-code abbreviations specified in the **Dialcodes** file for certain telephone numbers, you *must* enter the `\T` string as the token in those entries in the **Dialers** file.
blank	Leaving the *Token* field blank is the same as entering `\D`, so a blank is usually sufficient as a token if you have included complete telephone numbers in the **/usr/lib/uucp/Systems** file.

If the *Token* field does not represent a telephone number, enter the predefined string necessary to reach the dialer.

Entries for Hardwired Connections

In general, each entry for a hardwired connection consists of two lines. The first line specifies the port (line) that the BNU command uses to connect to the remote system. The second line specifies the remote system. However, if the two systems use a permanent virtual circuit connection, the entry is a single line in the **Devices** file.

To set up a hardwired connection specifying a port and a remote system, make a two-line entry as follows:

1. Enter the keyword `Direct`, with an uppercase `D`, in the *Type* field in the first line of the entry.
2. Enter the name of the remote system to which you want to connect the local computer over the hardwired line in the *Type* field in the second line of the entry.

3. Enter the device name appropriate for the hardwired connection used at your site in the *Line* field in both lines of the entry.
4. Enter a – (hyphen) for a placeholder in the *Line2* field in both lines of the entry.
5. Enter the transmission rate appropriate for the hardwired connection used at your site in the *Class* field in both lines of the entry.
6. Enter `direct` (all lowercase) in the *Dialer-Token Pairs* field in both lines of the entry.

To set up a hardwired connection between two systems that use a permanent virtual circuit connection, make a one-line entry as follows:

1. Enter the name of the remote system in the *Type* field.
2. Enter the name of the permanent virtual circuit connection in the *Line* field.
3. Enter a – (hyphen) for a placeholder in the *Line2* field.
4. Enter the transmission rate appropriate for the hardwired connection used at your site in the *Class* field.
5. Enter `direct` (all lowercase) in the *Dialer-Token Pairs* field.

Continue adding entries to the **Devices** file until you have listed each hardwired device connecting the local system to a remote system.

Entries for Autodialer Connections

In telephone-connection entries, the *Type* field is specified as an ACU. You should type ACU as the *Type* field entry in all remote connections established over a phone line. To set up **Device** file entries for autodialer connections, make a one-line entry for each modem as follows:

1. Enter `ACU` in the *Type* field.
2. The *Line* field contains the name of the device that is attached to the modem. Enter the device name appropriate for your site.
3. Enter a – (hyphen) as a placeholder in the *Line2* field, unless the autodialer is a standard 801 dialer. If the autodialer is a standard 801 dialer, enter `801`.
4. In the *Class* field, enter the baud rate appropriate for your modem and line (this can be 300, 1200, 2400, or higher, depending on the modem) or the class of your modem (for example, D2400).

   Note: If the modem can be used at more than one specific rate, make a separate entry in the **Devices** file for each rate. If the modem can be used at any rate, enter the word `Any` in the *Class* field.
5. Enter the name of the modem as the *Dialer* field entry in the *Dialer-Token Pair* field. If you are planning to include complete phone numbers in the **/usr/lib/uucp/Systems** file, leave the *Token* field blank. (A blank instructs the BNU program to use the default `\D` token.) If you are planning to use dialing-code abbreviations specified in the **/usr/lib/uucp/Dialcodes** file, enter the token `\T` .

Continue adding entries to the **Devices** file until you have listed each connection between the local system and a remote system that uses a telephone line and a modem.

Entry for Use with TCP/IP

If your site is using the TCP/IP system, include the relevant TCP/IP entry in the **Devices** file. To set up the file for use with the TCP/IP system, enter a line in the **Devices** file as follows:

1. Enter `TCP` in the *Type* field.
2. Enter hyphens in the *Line*, *Line2*, and *Class* fields.
3. Enter `TCP` as the *Dialer* field entry and leave the *Token* field blank.

Examples

Setting Up Entries for Hardwired Connections

1. To set up a **Device** file entry specifying a port and a remote system, make an entry as follows:

```
Direct tty1 - 1200 direct

zeus tty1 - 1200 direct
```

The *Type* field lists `Direct` (for a direct connection) in the first part and `zeus` (the name of the remote system) in the second part. The local system is connected to system `zeus` by way of device `tty1`, which is listed in the *Line* field in both parts of the example.

The *Line2* field contains actual data only when the entry specifies a certain type of telephone connection. A - (hyphen) is used as a placeholder in other types of connections, as is the case in this example. This device transmits at a rate of 1200 bps, which is listed in the *Class* field in both parts of the example. The word `direct` in the *Dialer* field portion of the *Dialer-Token Pair* field indicates that this is a direct connection.

Setting Up Entries for Autodialer Connections

2. For a standard Hayes modem that can be used at only one baud rate, make an entry as follows:

```
ACU tty2 - 1200 hayes
```

The *Type* field is specified as `ACU`. The *Line* field is specified with the device name `tty2`. Because this modem is not an 801 dialer, a - (hyphen) is used as a placeholder in the *Line2* field. The *Class* field entry is a transmission rate of `1200` baud. The *Dialer* field part of the *Dialer-Token Pair* field is specified as a `hayes` modem, and the *Token* field part is left blank.

3. To specify a standard Hayes modem that can be used at different baud rates, make an entry as follows:

```
ACU tty3 - 1200 hayes
ACU tty3 - 300 hayes
```

These two lines specify the same modem, a `hayes`, which can be used at either `1200` or `300` baud, as specified in the *Class* field. The modem is connected to a device named `tty3` (the *Line* field), and the *Line2* field contains the - (hyphen) placeholder. The *Dialer* field part of the *Dialer-Token Pair* field is specified as a `hayes` modem, and the *Token* field is left blank.

4. To specify a standard Hayes modem that can be used at any baud rate, make an entry as follows:

```
ACU tty2 - Any hayes
```

These two lines specify a hayes modem that can be used at any baud rate, as specified by the word Any entered in the *Class* field. Note that the word Any must be in with an uppercase A.

5. To specify a connection using a standard 801 dialer, make an entry as follows:

```
ACU tty4 tty5 1200 801
ACU tty6 tty7 300 801
```

In these entries, the ACU entries are connected to devices named tty4 and tty6, specified in the *Line* field. In both cases, there is an entry in the *Line2* field because a standard 801 autodialer is specified in the *Dialer-Token Pair* field. Because 801 is specified as the dialer in these two examples, the *Line2* field must contain the device names of the 801 ACUs. The *Class* field entry specifies a transmission rate of 1200 baud for the first example and 300 for the second. The *Token* field part of the *Dialer-Token Pair* field is blank.

Setting up the Entry for Use with TCP/IP

6. If your site is using the TCP/IP system, enter the following in the **Devices** file:

```
TCP - - - TCP
```

TCP is specified in the *Type* field. Hyphens are used as placeholders in the *Line*, *Line2*, and *Class* fields. TCP is specified as the *Dialer* field entry, with the *Token* entry left blank.

Setting Up Entries for Both Local and Remote Systems

The following examples illustrate the entries needed in the **Devices** file for both local and remote systems in order for the two systems to communicate using the BNU program.

7. To configure a hardwired connection, note the following information.

The following entries configure local and remote **Devices** files for a hardwired connection between systems zeus and hera, where zeus is considered the local system and hera remote system. The hardwired device on system zeus is tty1; on system hera it is tty2.

The **Devices** file on system zeus contains the following entry in order to connect to the remote system hera:

```
Direct tty1 - 1200 direct
hera tty1 - 1200 direct
```

The **Devices** file on system hera contains the following entry for communications with system zeus:

```
Direct tty2 - 1200 direct
zeus tty2 - 1200 direct
```

8. To configure a telephone connection, note the following information.

These files are set up to connect systems venus and merlin over a telephone line using modems. System venus is considered the local system, and system merlin is considered the remote system.

On both systems, the device tty1 is hooked to a hayes modem at 1200 baud. Both computers include partial phone numbers in their **/usr/lib/uucp/Systems** files and dialing codes in their **/usr/lib/uucp/Dialcodes** files.

The **Devices** file on system `venus` contains the following entry for the connection to system `merlin`:

```
ACU tty1 - 1200 hayes \T
```

The **Devices** file on system `merlin` contains the following entry for the connection to system `venus`:

```
ACU tty1 - 1200 hayes \T
```

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **Devices** file.
/usr/lib/uucp/Dialcodes file	Contains dialing code abbreviations.
/usr/lib/uucp/Dialers file	Specifies initial handshaking on a connection.
/usr/lib/uucp/Systems file	Describes accessible remote systems.

Related Information

The **uucpadm** command can be used to make entries in the **Devices** file.

The **uucico** daemon is used to establish and debug remote connections.

The **uuxqt** daemon is used to execute remote commands.

Commands used to establish remote connections are the **cu** command, **uucp** command, **uuto** command, and **uux** command.

How to Configure BNU and How to Configure BNU for Use with TCP/IP *Communication Concepts and Procedures*.

Understanding the BNU Configuration Files and Configuring BNU *Communication Concepts and Procedures*.

BNU Dialcodes File Format

Purpose

Contains the initial digits of telephone numbers used to establish remote connections over a phone line.

Description

The **/usr/lib/uucp/Dialcodes** file contains the initial digits of telephone numbers used by the Basic Networking Utilities (BNU) program to establish remote connections over a phone line. The **Dialcodes** file simplifies entries in the **/usr/lib/uucp/Systems** file for sites where a number of device phone numbers have the same prefix.

If users at your site communicate regularly by way of telephone lines and modems to multiple systems all located at the same remote site, or to multiple systems located at different remote sites, use the dial-code abbreviations in the **/usr/lib/uucp/Systems** file rather than entering the complete phone number of each remote modem in that file.

The **Dialcodes** file contains dial-code abbreviations and partial phone numbers that complete the telephone entries in the **/usr/lib/uucp/Systems** file. Entries in the **Dialcodes** file contain an alphabetic prefix attached to a partial phone number that may include the following information in the order listed:

- Codes for an outside line
- Long-distance access codes
- A 1 (one) plus the area code (if the modem is out of the local area)
- The three-digit exchange number.

The relevant alphabetic prefix (representing the partial phone number), together with the remaining four digits of that number, is then entered in the *Phone* field in the **/usr/lib/uucp/Systems** file.

Following is the form of an entry in a **Dialcodes** file:

DialCodeAbbreviation DialingSequence

The *DialCodeAbbreviation* part of the entry is an alphabetic prefix containing up to 8 letters, established when setting up the dialing-code listing. The *DialingSequence* is composed of all the digits in the number that precede the actual four-digit phone number.

Note: If your site uses only a relatively small number of telephone connections to remote systems, include the complete phone numbers of the remote modems in the **/usr/lib/uucp/Systems** file rather than use dial-code abbreviations.

Enter each prefix *only once* in the **Dialcodes** file. When you have set up a dial-code abbreviation, use that prefix in all relevant entries in the **/usr/lib/uucp/Systems** file.

Only someone with root user authority can edit the **Dialcodes** file, which is owned by the **uucp** program login ID.

Example

1. The **Dialcodes** file on system `venus` contains the following dial-code prefix for use with a number in the **/usr/lib/uucp/Systems** file:

   ```
   local 9=445
   ```

   The **Systems** file on system `venus` contains the following entry for system `zeus`, including a phone number and a dialing prefix:

   ```
   zeus Any ACU 1200 local8784 in:—in: uzeus word: thunder
   ```

   When BNU on system `venus` dials system `zeus`, BNU uses the expanded telephone number `9=4458784`.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **Dialcodes** file.
/usr/lib/uucp/Devices file	Contains information about available devices.
/usr/lib/uucp/Dialers file	Specifies initial handshaking on a connection.
/usr/lib/uucp/Systems file	Describes accessible remote systems.

Related Information

Understanding the BNU Configuration Files and Configuring BNU in *Communication Concepts and Procedures.*

How to Configure BNU in *Communication Concepts and Procedures.*

The **uucpadm** command can be used to make entries in the **Dialcodes** file.

Commands used to establish remote connections are the **cu** command, **tip** command, **uucp** command, **uuto** command, and **uux** command.

BNU Dialers File Format

Purpose

Lists modems used for Basic Networking Utilities (BNU) remote communications links.

Description

The **/usr/lib/uucp/Dialers** file lists the modems used by the Basic Networking Utilities (BNU) program and specifies the initial handshaking necessary to establish remote communications links. Handshaking is a series of expect-send sequences that specify the initial communications that occur on a link before it is ready to send or receive data. Using the handshaking, the local and remote systems confirm that they are compatible and configured to transfer data.

The **Dialers** file contains an entry for each autodialer that is included in the **/usr/lib/uucp/Devices** file. It also contains entries specifying no handshaking for direct hardware links (the direct entry) and TCP/IP links (the TCP entry). The first field of the **Dialers** file, which specifies the dialer, is matched to the fifth field of the **Devices** file, the *Dialer-Token Pair* field, to determine handshaking when making a connection.

Note: Only someone with root user authority can edit the **Dialers** file, which is owned by the **uucp** login ID.

Fields in the Dialers File

Every modem is listed on a line by itself in the **Dialers** file. Each line consists of three groups of information: the *Dialer Name* field, the *Dial Tone and Wait Characters* field, and the *Handshaking* field.

Dialer Name **Field**

The first field in the **Dialers** file, the *Dialer Name* field, specifies the type of autodialer (modem) used in the connection. It matches the fifth field, the *Dialer-Token Pair* field, in the **/usr/lib/uucp/Devices** file. When a particular device is used to make a connection, BNU uses the *Dialer-Token Pair* field in the **Devices** file to find the handshaking entry in the **Dialers** file.

If your system has direct hardware connections to one or more remote systems, include an entry with a *Dialer Name* of **direct**. Similarly, if your system uses TCP/IP to connect to one or more other systems, include an entry with a *DialerName* of **TCP**. These entries correspond, respectively, to the word `direct` and the word `TCP` in the *Dialer-Token Pairs* field of entries in the **/usr/lib/uucp/Devices** file. Omit the *Dial Tone and Wait Characters* field and the *Handshaking* field, since no handshaking is needed on these connections.

Dial Tone and Wait Characters **Field**

The second field, the *Dial Tone and Wait Characters* field, consists of two sets of two characters, for a total of four entries. These characters comprise a translation string. In the actual phone number of the remote modem, the first character in each string is mapped to the second character in that set.

Entry	Action
=,–,	Translate the telephone number. Any = (equal sign) represents *wait for dial tone* and any – (hyphen) represents *pause*.

""	Wait for nothing; continue with the rest of the string.

This field generally translates the = and – characters into whatever the dialer uses for *wait for dial tone* and *pause*.

For **direct** and **TCP** entries, omit this field.

Handshaking **Field**

The handshaking, or dialer negotiations, is an expect-send sequence of ASCII strings. This sequence is given in the *Handshaking* field, which comprises the remainder of the entry. This string is generally used to pass telephone numbers to a modem, or to make a connection to another system on the same data switch as the local system. The string tells the **cu** or **ct** program or the **uucico** daemon the sequence of characters to use to dial out on a particular type of modem. If the connection succeeds, the line in the **Dialers** file is interpreted to perform the dialer negotiations.

The handshaking characters include entries such as `\d` to specify a delay, `\p` for a pause, `\r` for a carriage return, and `\c` for a new line. To determine the appropriate entries in the handshaking string, refer to the documentation that accompanied the modems that you are including in the **Dialers** file and to the list of expect-send sequences given in the **/usr/lib/uucp/Systems** file format.

For **direct** and **TCP** entries, omit this field.

Examples

Setting Up Entries in the Dialers File

1. The following example lists several entries in a typical **Dialers** file:

```
hayes =,–, "" \dAT\r\c OK \pATDT\T\r\c CONNECT
penril =W–P "" \d > s\p9\c )–W\p\r\ds\p9\c–) y/c : \E\TP > 9\c OK
ventel =&–% "" \r\p \r\p–\r\p–$ <K\D%%\r>\c ONLINE!
vadic =K–K "" \005\p *–\005\p–* D\p BER? \E\D\e \r\c LINE
direct
TCP
```

   Note: In the **Dialers** file, each entry must be entirely on one line.

   Notice that the next-to-last entry in the preceding example consists only of the word `direct`. This entry indicates that hardwired connections do not require any handshaking. Similarly, the last entry, `TCP`, indicates that TCP/IP connections require no handshaking.

2. The following example interprets the first line in the preceding **Dialers** file. This is a standard entry that may be included in your **Dialers** file with modifications for use at your site.

```
hayes =,–, "" \dAT\r\c OK \pATDT\T\r\c CONNECT
```

   The first two sequences (=,– and "") comprise the *Dial Tone and Wait Characters* field. The remaining strings comprise the *Handshaking* field. Following is an explanation of how each entry affects the action of the dialer.

Entry	Action
=,–,	Translate the telephone number. Any = (equal sign) represents *wait for dial tone* and any – (hyphen) represents *pause*.
""	Wait for nothing; continue with the rest of the string.
\dAT	Delay, then send `AT` (the Hayes Attention prefix).

`\r\c`	Send a carriage return (`r`) followed by a new line (`c`).
`OK`	Wait for `OK` from the remote modem, signaling that the first part of the string has executed.
`\pATDT`	Pause (`p`), then send `ATDT`. `AT` is again the Hayes Attention prefix, `D` represents a dialing signal, and `T` represents a touch-tone dial tone.
`\T`	Send the telephone number, which is specified in the **/usr/lib/uucp/Systems** file, with dial-code translation from the **/usr/lib/uucp/Dialcodes** file.
`\r\c`	Send a carriage return and a new line following the number.
`CONNECT`	Wait for `CONNECT` from the remote modem, signaling that the modems are connected at the baud rate specified in the **/usr/lib/uucp/Devices** file.

Note: If you need to modify this example for use at your site and are unsure about the appropriate entries in the handshaking string, refer to the documentation that accompanied the modems you are including in the **Dialers** file.

Setting Up the Direct Entry

3. If your BNU configuration includes hardwired connections, the **Dialers** file must contain a **direct** entry, as follows:

   ```
   direct
   ```

   This entry indicates that hardwired connections do not require any handshaking. It corresponds to the word `direct` in the *Dialer-Token Pairs* field of entries for hardwired devices in the **/usr/lib/uucp/Devices** file.

Setting Up the TCP/IP Entry

4. If your BNU configuration includes TCP/IP connections, the **Dialers** file must contain a **TCP** entry, as follows:

   ```
   TCP
   ```

   This entry indicates that TCP/IP connections do not require any handshaking. It corresponds to the word `TCP` in the *Dialer-Token Pairs* field of entries for TCP/IP connections in the **/usr/lib/uucp/Devices** file.

Setting Up Entries for Both Local and Remote Systems

5. The following example illustrates the entries needed in the **Dialers** file to correspond to entries in the **/usr/lib/uucp/Devices** file for both local and remote systems so that the two systems can communicate using the BNU program.

   These files are set up to connect systems `venus` and `merlin` over a telephone line using modems. System `venus` is considered the local system, and system `merlin` is considered the remote system. On both systems, the device `tty1` is hooked to a `hayes` modem at `1200` baud.

- The **/usr/lib/uucp/Devices** file on system `venus` contains the following entry for the connection to remote system `merlin`:

  ```
  ACU tty1 - 1200 hayes
  ```

- The **Dialers** file on system `venus` contains the following entry for its modem:

  ```
  hayes =,-, "" \dAT\r\c OK \pATDT\T\r\c CONNECT
  ```

- The **/usr/lib/uucp/Devices** file on system `merlin` contains the following entry for the connection to system `venus`:

```
ACU tty1 - 1200 hayes
```

- The **Dialers** file on system `merlin` contains the following entry for its modem:

```
hayes =,-, "" \dAT\r\c OK \pATDT\T\r\c CONNECT
```

Troubleshooting Connection Problems

6. When establishing a connection between a local and a remote system using a telephone line and modem, the BNU program consults the **Dialers** file. (The BNU program also checks the **/usr/lib/uucp/Systems** file to make sure it contains a listing for the specified remote computer.) If users report a faulty connection, use the **uucico** command to debug the connection problem. For example, if users are experiencing difficulties connecting to remote system `venus`, issue the following command:

   ```
   /usr/lib/uucp/uucico -r1 -svenus -x9
   ```

   where `-r1` specifies the server mode, `-svenus` the name of the remote system to which you are trying to connect, and `-x9` the debug level that produces the most detailed debugging information.

   Expect-send debugging output produced by the **uucico** command can come either from information in the **Dialers** file or from information in the **/usr/lib/uucp/Systems** file. If the relevant line in the **Dialers** file is not set up correctly for the specified modem, the BNU program will probably display the following error message:

   ```
   DIALER SCRIPT FAILED
   ```

   If the dialer script fails, verify the following:

   - Make sure that both the local and the remote modems are turned on, that they are both set up correctly, and that the telephone number of the remote modem is correct.
   - Check the **Dialers** file and make sure the information is correctly specified for the local modem. If possible, also check the **Dialers** file on the remote system.
   - Check the documentation that came with your modem to make sure you have used the correct expect-send sequence characters in the **Dialers** file.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **Dialers** file.
/usr/lib/uucp/Devices file	Contains information about available devices.
/usr/lib/uucp/Dialcodes file	Contains dialing code abbreviations.
/usr/lib/uucp/Systems file	Describes accessible remote systems.

Related Information

How to Configure BNU in *Communication Concepts and Procedures*.

The daemon used to establish and debug remote connections is the **uucico** daemon. The procedures How to Monitor a BNU Remote Connection and How to Use the **uucico** Daemon to Debug BNU Login Failures in *Communication Concepts and Procedures* suggest steps to follow when using the **uucico** daemon for debugging.

Commands used to establish remote connections are the **ct** command and **cu** command.

Commands used to debug remote connections are the **uutry** command, **Uutry** command, and **uukick** command.

Understanding the BNU Configuration Files and Configuring BNU in *Communication Concepts and Procedures*.

BNU Maxuuscheds File Format

Purpose

Limits the number of instances of the **uusched** and **uucico** daemons that can run simultaneously.

Description

The **/usr/lib/uucp/Maxuuscheds** file limits the number of instances of the Basic Networking Utilities (BNU) **uusched** daemons that can run simultaneously. Since each instance of the **uusched** daemon is associated with one instance of the **uucico** daemon, the file limits the instances of the **uucico** daemon in a similar way. This file is used in conjunction with the lock files in the **/etc/locks** directory to determine the number of systems currently being polled. Use this file to help manage system resources and load averages.

The **Maxuuscheds** file contains an ASCII number that can be changed in order to meet the needs of your installation; the default is 2. The larger the number, the greater the potential load on the local system. In any case, the limit should always be less than the number of outgoing lines used by BNU.

The **Maxuuscheds** file requires neither configuration nor maintenance, unless the system on which it is installed is contacted frequently and heavily by users on remote systems.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/etc/locks directory	Contains lock files that prevent multiple uses of devices and multiple calls to systems.
/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **Maxuuscheds** file.

Related Information

Understanding the BNU Configuration Files and Configuring BNU in *Communication Concepts and Procedures*.

The **uucico** daemon contacts remote systems. The **uusched** daemon schedules contacts with remote systems.

BNU Maxuuxqts File Format

Purpose

Limits the number of instances of the BNU **uuxqt** daemon that can run simultaneously on the local system.

Description

The **/usr/lib/uucp/Maxuuxqts** file limits the number of instances of the Basic Networking Utilities (BNU) **uuxqt** daemon that can run simultaneously on the local system, thus limiting the number of commands from remote systems that can be running at any one time.

This file contains an ASCII number that can be changed in order to meet the needs of your installation; the default value is 2. In general, the larger the number, the greater the potential load on the local system.

The **Maxuuxqts** file requires neither configuration nor maintenance, unless the system on which it is installed is used frequently and heavily by users on remote systems.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

File

/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **Maxuuxqts** file.

Related Information

Understanding the BNU Configuration Files and Configuring BNU in *Communication Concepts and Procedures*.

The **uuxqt** daemon executes commands from remote systems.

BNU Permissions File Format

Purpose

Specifies BNU permissions for remote systems that call or are called by the local system.

Description

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that use the Basic Networking Utilities (BNU) program to communicate with the local system. The **Permissions** file contains an entry for each system that the local system contacts using BNU. These entries correspond to entries in the **/usr/lib/uucp/Systems** file. The **Permissions** file also contains an entry for each login ID that remote systems are permitted to use when using BNU to log into the local system.

Entries in the **Permissions** file specify the following:

- The login ID for a remote system
- The circumstances under which a remote system is allowed to send files to and receive files from the local system
- The commands a remote system is permitted to execute on the local system.

The access permissions that you set in a **Permissions** file affect remote systems as a whole. They do *not* pertain to individual users who work on those remote systems. Permissions limiting **uucico** and **uuxqt** daemon activities restrict the BNU access to a local system by *all* users on a specified remote system. Restricted access includes the BNU or UUCP administrator of that remote system and individuals with root user privileges (unless you have set up a special login and password for such users). Conversely, less restrictive permissions allow more generous BNU access to *all* users on the specified remote system.

The default permissions for sending and receiving files and executing commands are very restrictive. However, the file also provides options that enable you to change these defaults if you want to allow remote systems to have less restricted access to the local system.

Warning: Any access permissions set in the **Permissions** file affect all BNU communications, including those made through the mail facility or over a TCP/IP connection.

Warning: Entries in a **Permissions** file do *not* affect a remote-system user with a valid login on a specified local system. Such users can use remote login commands (such as **cu**, **ct**, **tn**, or **tip**) to connect to, log in on, and use a system regardless of the restrictions you set up in the local **Permissions** file. A user with a valid login ID is subject only to the permission codes established for that user's user ID (UID) and group ID (GID).

Each entry in a **Permissions** file is a logical line. If an entry is too long to fit on the screen, make the last character in that physical line a \ (backslash), which indicates continuation, and then type the remainder of the entry on the next physical line.

Each logical line contains a required entry specifying a login ID (LOGNAME entry) or the name of a remote system (MACHINE entry), followed by optional Option/Value pairs separated by either spaces or tabs. Both the LOGNAME and MACHINE entries and the Option/Value pairs are composed of name/value pairs. Name/value pairs consist of the name of the entry or option followed by = (an equal sign) and the value of the entry or option, with no spaces allowed within the pair.

The **Permissions** file can also contain comment lines and blank lines. Comment lines begin with a # (pound sign) and occupy the entire physical line. Blank lines are ignored.

Note: Only someone with root user authority can edit the **Permissions** file, which is owned by the **uucp** login ID.

LOGNAME and MACHINE Entries

The **Permissions** file contains two types of required entries, the LOGNAME entry and the MACHINE entry. The LOGNAME entry specifies the login IDs and access permissions for remote systems that are allowed to contact the local system. The MACHINE entry specifies the names and access permissions for the remote systems that the local system can contact.

Both LOGNAME and MACHINE entries specify *what the remote system can do on the local system*. However, LOGNAME entries take effect when a remote system contacts the local system, while MACHINE entries take effect when the local system contacts a remote system. The permissions given to the remote system in the two types of entries can be the same or different. For example, since it is relatively easy for an unknown system to call you and log in on your system, you may want to restrict permissions in LOGNAME entries.

If system `hera` contacts system `zeus` and logs in as `uhera`, the `LOGNAME=uhera` entry in the **Permissions** file on `zeus` controls what actions system `hera` can take on system `zeus`. If system `zeus` contacts system `hera`, the `MACHINE=hera` entry in the **Permissions** file on `zeus` controls what actions system `hera` can take on system `zeus`.

The formats for the **Permissions** file entries are:

Entry	Description
LOGNAME	Specifies the login IDs that remote systems can use when calling the local system. **LOGNAME=***LoginID*[**:***LoginID* . . .] [*Option*=*Value* . . .] Remote systems log in with one of the IDs listed in the *LoginID* list. While logged in with that ID, the remote system has the permissions specified in the *Option*=*Value* list. The remote system that is calling must be listed in the **/usr/lib/uucp/Systems** file on the local system. **Note:** All login IDs used by calling remote systems must be in a LOGNAME entry in the **Permissions** file. There can be *only one* LOGNAME entry in the **Permissions** file for each login ID.
MACHINE	Specifies the systems that the local system can contact **MACHINE=***SystemName*[**:***SystemName* . . .] [*Option*=*Value* . . .] The local system can contact the systems listed. When contacted, the remote systems will be able to carry out operations on the local system according to the permissions specified in the *Option*=*Value* list.

MACHINE=OTHER [*Option=Value . . .*]

Systems not specifically listed in another MACHINE entry can contact the local system with the permissions specified in the *Option=Value* list.

Note: The local system cannot call any remote system that is not listed by name in a MACHINE entry, unless there is a MACHINE=OTHER entry in the **Permissions** file on the local system.

Combined LOGNAME and MACHINE Entries

A LOGNAME and a MACHINE entry can be combined into a single entry when both parts include the same options.

Format of Option/Value Pairs

Several option/value pairs can be used with the LOGNAME and MACHINE entries. The default permissions are restrictive, but can be changed with one or more of the option/value pairs. These options allow different remote systems different types of access to the local system when using the BNU file transport and command execution programs. The option/value pairs and their formats are:

Option **Description**

REQUEST Specifies when a remote system can request file transfers. The default is REQUEST=no.

REQUEST=yes When it contacts the local system the remote system can request that files be transferred.

REQUEST=no The remote system must wait for a contact from the local system before the files can be transferred.

SENDFILES Specifies when files can be transferred to a remote system. Used only in the LOGNAME entry. The default is SENDFILES=call.

SENDFILES=yes
Files can be sent to the remote system immediately after the remote system finishes its operations on the local system.

SENDFILES=call
Files cannot be sent to the remote system until the local system initiates a contact to the remote system.

READ Designates the directories in which reside files that the remote system can read.

READ=*PathName*[**:***PathName . . .*]
The remote system can read any file residing in the *PathName* directory (or subdirectories of the *PathName* directory) if the file permissions include read rights for the others group.

WRITE Designates the directories containing files to which the remote system can write.

WRITE=*PathName*[**:***PathName* . . .]
The remote system can write to any file residing in the *PathName* directory (or subdirectories of the *PathName* directory) if the file permissions include write rights for the others group.

NOREAD Specifies exceptions to the READ option.

NOREAD=*PathName*[**:***PathName* . . .]
The remote system cannot read any file residing in the *PathName* directory (or subdirectories of the *PathName* directory), even if the parent directory of the *PathName* directory is accessible to the remote system.

NOWRITE Specifies exceptions to the WRITE option.

NOWRITE=*PathName*[**:***PathName* . . .]
The remote system cannot write to any file residing in the *PathName* directory (or subdirectories of the *PathName* directory), even if the parent directory of the *PathName* directory is accessible to the remote system.

COMMANDS Specifies commands that remote systems can execute. Used only in the MACHINE entry. The default is the **rmail** command. Can include the path name of the command.

COMMANDS=*CommandName*[**:***CommandName* . . .]
The BNU program allows the remote system to execute only the commands specified in the *CommandName* list.

COMMANDS=ALL
The BNU program allows the remote system to execute all AIX commands available on the local system.

Warning: The COMMANDS=ALL option/value pair can jeopardize the security of your system. Use it with extreme care.

VALIDATE Requires a unique login and password from a remote system. Used only in a LOGNAME entry. If a LOGNAME entry contains the VALIDATE option, the remote systems listed in the VALIDATE option cannot log into the remote system with any other login ID.

The VALIDATE option is usually used in conjunction with the COMMANDS=ALL option, or when the permissions granted with the LOGNAME entry include potentially dangerous access. The format for the VALIDATE option is as follows:

VALIDATE=*SystemName*[**:***SystemName* . . .]
Validation applies to each system listed in the *SystemName* list.

CALLBACK Specifies whether the remote system can initiate file transfers to the local system. Used only in a LOGNAME entry. The default is CALLBACK=no.

CALLBACK=yes
The local system must contact the remote system before the remote system can begin file transfers to the local system.

CALLBACK=no
The BNU program allows the remote system to contact the local system and transfer files without waiting for the local system to initiate the transaction.

If two systems both include the CALLBACK=yes option in their respective **Permissions** files, they cannot communicate with each other using BNU.

LOGNAME Entry

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. The **Permissions** file contains a LOGNAME entry for each login ID that remote systems are permitted to use to log into the local system.

A LOGNAME entry specifies one or more login IDs for remote systems permitted to log into the local system to conduct **uucico** and **uuxqt** daemon transactions, plus the access permissions for those remote systems. The login ID can be any valid login name. The LOGNAME entry specifies permissions for the remote system *when it contacts the local system*. To specify permissions for the remote system when it is called by the local system, use a MACHINE entry.

Following is the format of a LOGNAME entry:

LOGNAME=*LoginID*[**:***LoginID* . . .] [*Option*=*Value* . . .]

To specify more than one login ID with the same option/value pairs, list them in the same LOGNAME entry, separated by colons but without spaces. To specify multiple login IDs with different option/value pairs, list them in separate LOGNAME entries.

The most restrictive LOGNAME entry is an entry without any option/value pairs, which means that the remote system's access to the local system is defined by the following default permissions:

- The remote system cannot ask to receive any queued files from the local system.
- The local system cannot send queued work to the calling remote system when the remote system has completed its current operations. Instead, the queued work can be sent only when the local system contacts the remote system.
- The remote system cannot send files to (write) or transfer files from (read) any location except the BNU public directory (**/usr/spool/uucppublic/***SystemName*) on the local system.
- Users on the remote system can execute only the default commands on the local system. (The default command set includes only the **rmail** command, which users implicitly execute by issuing the **mail** command.)

To override these defaults, include option/value pairs in the LOGNAME entry. The available options are:

REQUEST	COMMANDS
SENDFILES	VALIDATE
READ, WRITE	CALLBACK
NOREAD, NOWRITE	

These options allow different remote systems different types of access to the local system when using the BNU file transport and command execution programs.

Note: A login ID can appear in only one LOGNAME entry. If there is a single entry for a login ID, that entry alone is sufficient for all remote systems using that login ID.

A LOGNAME entry that contains a restricted login ID is generally sufficient for **uucico** and **uuxqt** daemon transactions between local and remote systems at most sites. However, alternate login IDs should be used if certain remote systems require different types of permissions when accessing the local system.

Warning: Allowing remote systems to login to the local system with the **uucp** login ID seriously jeopardizes the security of your system. Remote systems logged in with the **uucp** ID can display and possibly modify (depending on the other permissions specified in the LOGNAME entry) the local **Systems** and **Permissions** files. It is strongly recommended that you create other BNU login IDs for remote systems and reserve the **uucp** login ID for the person responsible for administering BNU on the local system. For the best security, each remote system that contacts the local system should a have unique login ID with a unique UID.

MACHINE Entry

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that the local system communicates with using the Basic Networking Utilities (BNU) program. Typically, the **Permissions** file contains a MACHINE entry for each remote system that the local system is permitted to contact.

The MACHINE entry contains the names of the remote systems that the local system is permitted to call and the access permissions for those remote systems. The access permissions specified in the MACHINE entry affect the remote system's access to the local system *when the local system contacts the remote system.* To specify access permissions when the remote system contacts the local system, use a LOGNAME entry.

Following is the format of a MACHINE entry:

MACHINE=*SystemName*[**:***SystemName* . . .] [*Option*=*Value* . . .]

Or

MACHINE=OTHER [*Option*=*Value* . . .]

The most restrictive type of MACHINE entry, which uses the default permissions, is as follows:

MACHINE=*SystemName*[**:***SystemName* . . .]

Notice that the system names are separated by a colon, and that the entry includes no spaces or tab characters. In this entry, there are no option/value pairs, indicating that remote system access to the local system is defined by the following default permissions:

- The remote system cannot ask to receive any local system files queued to run on the calling remote system.
- The remote system cannot access (read) any files except those in the public directory on the local system.
- The remote system can send (write) files only to the local public directory.
- The remote system can execute only those commands in the default command set on the local system.

To override these defaults, include option/value pairs in the MACHINE entry. The available options are:

REQUEST COMMANDS
SENDFILES VALIDATE
READ, WRITE CALLBACK
NOREAD, NOWRITE

These options allow different remote systems different types of access to the local system when using the BNU file transport and command execution programs.

The *SystemName* list in a MACHINE entry may include a number of different remote systems.

A MACHINE entry can also be in the following format:

MACHINE=OTHER [*Option=Value . . .*]

The MACHINE=OTHER format, in which the word OTHER represents a system name, sets up access permissions for remote systems not explicitly specified in the existing MACHINE entries in a **Permissions** file.

The MACHINE=OTHER entry is useful in the following circumstances:

- When your installation includes a large number of remote systems that the local system regularly contacts to conduct **uucico** and **uuxqt** daemon transactions
- When it is occasionally necessary to change the default command set specified in the COMMANDS option in the MACHINE entry.

Rather than creating separate MACHINE entries for each of a large group of remote systems, set up one MACHINE=OTHER entry that includes the appropriate commands specified in a COMMANDS option entry. Then, when it becomes necessary to change the default command set, change the list of commands in only one entry rather than in numerous entries. Usually, a MACHINE=OTHER entry also specifies more restrictive option values for the unidentified remote systems.

CALLBACK Option

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. LOGNAME entries specify permissions in effect when a remote system contacts the local system.

The CALLBACK option, which can be included in LOGNAME entries, specifies that no file transfer transactions will occur until the local system contacts the remote system that is attempting to establish a connection.

The format of the CALLBACK option is either

CALLBACK=no

Or

CALLBACK=yes

Warning: If two systems both include the CALLBACK=yes option in their respective **Permissions** files, they cannot communicate with each other using BNU.

The default value, CALLBACK=no, specifies that the remote system may contact the local system and begin transferring files without the local system initiating the operations.

For tighter security, specify that the local system must contact the remote system before that remote system may transfer any files to the local system. The CALLBACK=yes option implements this restriction.

If you include the CALLBACK=yes option in the LOGNAME entry for a login ID used by a particular remote system, you must also have a MACHINE entry for that system so that your system can call it back. Alternatively, you can have a MACHINE=OTHER entry to allow your system to call any remote system, including the one for which the CALLBACK=yes option is specified.

The default value, CALLBACK=no, is generally sufficient for most sites.

COMMANDS Option

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. MACHINE entries in the **Permissions** file specify permissions in effect when the local system contacts a remote system.

The COMMANDS option, which can be included only in a MACHINE entry, specifies the commands that the remote systems listed in that MACHINE entry can execute on the local system.

Warning: The COMMANDS option can jeopardize the security of your system. Use it with extreme care.

Following are the formats for this option:

COMMANDS=*CommandName*[**:***CommandName* . . .]

OR

COMMANDS=ALL

The default for this option is `COMMANDS=rmail:uucp`. Under the default, remote systems can run only the **rmail** command and the **uucp** command on the local system. (Users enter the **mail** command, which then calls the **rmail** command.)

The commands listed in the COMMANDS option in the MACHINE part of an entry in the **Permissions** file override the default.

You can also specify path names to those locations on the local system where commands issued by users on remote systems are stored. Specifying path names is useful when the default path of the **uuxqt** daemon does not include the directory where a command resides.

Note: The default path of the **uuxqt** daemon includes only the **/bin** and **/usr/bin** directories.

To allow a certain remote system to execute all available commands on the local system, use the COMMANDS=ALL format. This specifies that the command set available to the designated remote system includes all commands available to users on the local system.

Warning: The COMMANDS=ALL option/value pair can jeopardize the security of your system. Use it with extreme care.

NOREAD and NOWRITE Options

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. LOGNAME entries specify permissions in effect when a remote system contacts the local system. MACHINE entries specify permissions in effect when the local system contacts a remote system.

The NOREAD and NOWRITE options, which can be used in both LOGNAME and MACHINE entries, delineate exceptions to the READ and WRITE options by explicitly forbidding access by the remote system to directories and files on the local system.

The formats of these options follow:

NOREAD=*PathName*[**:***PathName* . . .]

NOWRITE=*PathName*[**:***PathName* . . .]

Note: The specifications you enter with the READ, WRITE, NOREAD, and NOWRITE options affect the security of your local system in terms of BNU transactions.

READ and WRITE Options

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. LOGNAME entries specify permissions in effect when a remote system contacts the local system. MACHINE entries specify permissions in effect when the local system contacts a remote system.

The READ and WRITE options, which can be used in both LOGNAME and MACHINE entries, specify the path names of directories that the **uucico** daemon can access when transferring files to or from the local system. The default location for both the READ and WRITE options is the **/usr/spool/uucppublic** directory (the BNU public directory) on the local system. The formats for these options follow:

READ=*PathName*[**:***PathName* . . .]

WRITE=*PathName*[**:***PathName* . . .]

The source file, destination file, or directory must be readable or writable for the other group for the BNU program to access it. Set these permissions with the **chmod** command. A user without root user authority can take away permissions granted by the READ and WRITE options, but that user cannot grant permissions that are denied by these options.

You can specify more than one path for **uucico** daemon activities.

If the READ and WRITE options are not present in the **Permissions** file, the BNU program transfers files only to the **/usr/spool/uucppublic** directory. However, if you specify path names in these options, enter the path name for every source and destination, including the **/usr/spool/uucppublic** directory if the remote system is to be permitted access to it.

Warning: The specifications you enter with the READ, WRITE, NOREAD, and NOWRITE options affect the security of your local system in terms of BNU transactions.

Warning: The subdirectories of directories specified in the READ and WRITE options can also be accessed by the remote system unless these subdirectories are forbidden with the NOREAD or NOWRITE options.

REQUEST Option

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. LOGNAME entries specify the permissions that are in effect when a remote system contacts the local system. MACHINE entries specify the permissions that are in effect when the local system contacts a remote system.

The REQUEST option can be used in both LOGNAME and MACHINE entries to enable a remote system to ask to receive any queued files containing work that users on the local system have requested to be executed on that remote system. The default is not to allow such requests.

When a remote system contacts the local system to transfer files or execute commands, that remote system might also request permission to receive any files queued on the local system for transfer to or execution on that remote system. The following format of the REQUEST option permits such requests:

REQUEST=yes

The default, which does not have to be entered, is REQUEST=no. This specifies that the remote system cannot ask to receive any work queued for it on the local system. In this case, the local system must contact the remote system before files and execute commands queued on the local system can be transmitted to the remote system.

Use the REQUEST=yes option in both LOGNAME and MACHINE entries in the **Permissions** file to allow remote-system users to transfer files to and execute commands a local system on demand. If security is a consideration at your site, restrict this access with the REQUEST=no option so that the local system retains control of file transfers and command executions initiated by remote systems.

Note: Entries in the **Permissions** file affect only BNU transactions. They do not affect remote-system users with valid logins on a local system.

SENDFILES Option

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. LOGNAME entries specify permissions in effect when a remote system contacts the local system. The **Permissions** file SENDFILES option can be used in a LOGNAME entry to specify when the local system can send queued work to the calling remote system.

The default is to allow the local system to transfer queued work to the remote system only when the local system contacts the remote system. However, when a remote system finishes transferring files to or executing commands on a local system, that local system may try to send queued work to the calling remote system immediately. To enable an immediate transfer, use the SENDFILES option as follows:

SENDFILES=yes

The SENDFILES=yes option allows the transfer of queued work from the local to the remote system once the remote system has completed its operations.

The default value, SENDFILES=call, specifies that local files queued to run on the remote system are sent only when the local system contacts the remote system. Security considerations at your site may require limiting access to the local system by a remote system by using the default value for this option.

Note: The SENDFILES option is ignored when it is included in a MACHINE entry.

Entries in the **Permissions** file affect only BNU transactions. They do not affect remote-system users with valid logins on a local system.

VALIDATE Option

The **/usr/lib/uucp/Permissions** file specifies access for remote systems that communicate with the local system using the Basic Networking Utilities (BNU) program. LOGNAME entries specify permissions in effect when a remote system contacts the local system. MACHINE entries specify permissions in effect when the local system contacts a remote system.

The VALIDATE option provides more security when it is necessary to include commands in the default command set that could potentially cause damage when executed by a remote system on a local system. Use this option, which can be specified only in a LOGNAME entry, in conjunction with a COMMANDS option in a related MACHINE entry.

Following is the format of the VALIDATE option:

VALIDATE=*SystemName*[:*SystemName* . . .]

The VALIDATE option verifies the identity of the calling remote system. Including this option in a LOGNAME entry means that the calling remote system must have a unique login ID and password for file transfers and command executions.

Note: This option is meaningful only when the login ID and password are protected. Giving a remote system a special login and password that provide unlimited file access and remote command-execution ability is equivalent to giving any user on that remote system a normal login and password on the local system, unless the special login and password are well-protected.

The VALIDATE option links a MACHINE entry, which includes a specified COMMANDS option, to a LOGNAME entry associated with a privileged login. The **uuxqt** daemon, which executes commands on the local system on behalf of users on a remote system, is not running while the remote system is logged in and therefore does not know which remote system sent the execution request.

Each remote system permitted to log in to a local system has its own spooling directory on that local system. Only the BNU file transport and command execution programs are allowed to write to these directories. For example, when the **uucico** daemon transfers execution files from the remote system `hera` to the local system `zeus`, it places these files in the `/usr/spool/uucppublic/hera` directory on system `zeus`.

Then, when the **uuxqt** daemon attempts to execute the specified commands, it determines the name of the calling remote system (`hera`) from the path name of the remote-system spooling directory (`/usr/spool/uucppublic/hera`). The daemon then checks for that name in a MACHINE entry in the **Permissions** file. The daemon also checks for the commands specified in the COMMANDS option in a MACHINE entry to determine whether the requested command can be executed on the local system.

Examples

Providing Default Access to Remote Systems

1. The following entry provides the default permissions to any system logging in as `uucp1`:

   ```
   LOGNAME=uucp1
   ```

2. The following entry provides the default permissions to systems `venus`, `apollo`, and `athena` when called by the local system:

   ```
   MACHINE=venus:apollo:athena
   ```

Providing Less Restricted Access to Remote Systems

3. The following LOGNAME entry allows remote system `merlin` to read and write to more directories than just the spool directory:

   ```
   LOGNAME=umerlin VALIDATE=merlin \
   READ=/ NOREAD=/etc:/usr/lib/uucp \
   WRITE=/u/merlin:/usr/spool/uucppublic
   ```

   A system logging in as user `umerlin` can read all directories *except* the **/usr/lib/uucp** and **/etc** directories, but can write only to the **/u/merlin** and public directories. Because the login name `umerlin` has access to more information than is standard, BNU validates the system before allowing `merlin` to log in.

4. The following example allows remote system `hera` unrestricted access to system `zeus`, and shows the relationship between the LOGNAME and MACHINE entries in a **Permissions** file:

   ```
   LOGNAME=uhera VALIDATE=hera REQUEST=yes SENDFILES=yes \
   READ=/ WRITE=/
   MACHINE=hera REQUEST=yes COMMANDS=ALL READ=/ WRITE=/
   ```

   The remote system `hera` may engage in the following **uucico** and **uuxqt** transactions with the local system `zeus`:

   - System `hera` may request that files be sent from system `zeus`, regardless of which system placed the call (`REQUEST=yes` appears in both entries);
   - System `zeus` may send files to system `hera` when system `hera` contacts system `zeus` (`SENDFILES=yes` in the LOGNAME entry);

- System `hera` may execute all available AIX commands on system `zeus` (`COMMANDS=ALL` in the MACHINE entry);
- System `hera` may read from and write to all directories and files under the **root** directory on system `zeus`, regardless of which system placed the call (`READ=/ WRITE=/` in both entries).

Because the entries provide system `hera` with relatively unrestricted access to system `zeus`, BNU validates the log name before permitting system `hera` to log in.

Warning: An entry like the one in the preceding example allows unrestricted access to the local system by the remote system listed in the MACHINE entry. Such an entry can jeopardize the security of your system. If security is a concern at your site, use such entries with extreme care.

Combining LOGNAME and MACHINE Entries

5. Following are LOGNAME and a MACHINE entries for system `hera`:

```
LOGNAME=uhera VALIDATE=hera REQUEST=yes SENDFILES=yes
MACHINE=hera REQUEST=yes COMMANDS=rmail:news:uucp
```

Since they have the same permissions and apply to the same remote system, these entries can be combined as follows:

```
LOGNAME=uhera VALIDATE=hera SENDFILES=yes REQUEST=yes \
MACHINE=hera COMMANDS=rmail:news:uucp
```

6. Similarly, LOGNAME and MACHINE entries used for more than one remote system can be combined if they have the same permissions. For example:

```
LOGNAME=uucp1 REQUEST=yes SENDFILES=yes
MACHINE=zeus:apollo:merlin REQUEST=yes COMMANDS=rmail:uucp
```

can be combined as:

```
LOGNAME=uucp1 REQUEST=yes SENDFILES=yes \
MACHINE=zeus:apollo:merlin COMMANDS=rmail:uucp
```

Either form of the entries allows systems `zeus`, `apollo`, and `merlin` the same permissions. They can:

- Log into the local system as `uucp1`
- Execute the **rmail** and **uucp** commands
- Request files from the local system, regardless of which system placed the call.

Allowing Access to Unnamed Systems

7. To allow your system to call systems which are not specified by name in a MACHINE entry, use a MACHINE=OTHER entry as follows:

```
MACHINE=OTHER COMMANDS=rmail
```

This entry allows your system to call any machine. The machine called will be able to request execution of the **rmail** command. Otherwise, the default permissions apply.

Permissions File Entries for Three Systems

8. The following examples show the **Permissions** files for three connected systems.

   On system venus:

```
LOGNAME=uhera MACHINE=hera \
READ=/ WRITE=/ COMMANDS=ALL \
NOREAD=/usr/secure:/usr/lib/uucp \
NOWRITE=/usr/secure:/usr/lib/uucp
SENDFILES=yes REQUEST=yes VALIDATE=hera
```

   On system hera:

```
LOGNAME=uvenus MACHINE=venus \
READ=/ WRITE=/ COMMANDS=rmail:who:lp:uucp \
SENDFILES=yes REQUEST=yes

LOGNAME=uucp1 MACHINE=OTHER \
REQUEST=yes SENDFILES=yes
```

   On system apollo:

```
LOGNAME=uhera MACHINE=hera \
READ=/usr/spool/uucppublic:/u/hera \
REQUEST=no SENDFILES=call
```

   Given these permissions, system hera logs into system venus as uhera. It can request or send files regardless of who initiated the call, and can read or write to all directories except **/usr/secure** and **/usr/lib/uucp**. It can execute any command. However, before system venus allows any system to log in as uhera, it checks to make sure that system is hera.

   System venus logs into system hera as uvenus. After it logs in, it can read or write to all directories on system hera and can request or send commands regardless of who initiated the call. It can execute the **rmail**, **who**, **lp**, and **uucp** commands only.

   System hera logs into system apollo as uhera. After it logs in, it can send files, but requests to receive files will be denied. It can read and write only from the public directory and the **/u/hera** directory, and can execute only the default list of commands.

   System apollo logs into system hera as uucp1, since it does not have a unique login ID on system hera. It can request and send files, regardless of who initiated the call. It can read and write only from the public directory (the default) and execute only the default list of commands.

   Note: The uucp1 login ID defined on system hera can be used by any remote system, not just by system apollo. In addition, the presence of the MACHINE=OTHER entry in this entry allows system hera to call machines not specified elsewhere in the **Permissions** file. If system hera calls an unknown machine, the permissions in the MACHINE=OTHER entry take effect.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **Permissions** file.
/usr/lib/uucp/Systems file	Describes accessible remote systems.
/usr/spool/uucppublic directory	Contains files that have been transferred.

Related Information

The **Permissions** file LOGNAME and MACHINE entries can be used to specify permissions for remote systems.

The following **Permissions** file options can be used to modify default permissions specified with the LOGNAME and MACHINE options: the REQUEST option, COMMANDS option, SENDFILES option, VALIDATE option, READ and WRITE options, NOREAD and NOWRITE options, CALLBACK option.

The **uucico** daemon and **uuxqt** daemon read the **Permissions** file.

The **uucheck** command, **rmail** command, **mail** command, and **chmod** command.

The **uucpadm** command can be used to make entries in the **Permissions** file.

How to Configure BNU and How to Configure BNU for use with TCP/IP in *Communication Concepts and Procedures* discuss entries in the **Permissions** file.

Understanding BNU Security and Understanding the BNU Configuration File in *Communication Concepts and Procedures*.

Configuring BNU in *Communication Concepts and Procedures*.

BNU Poll File Format

Purpose

Specifies when the BNU program should poll remote systems.

Description

The **/usr/lib/uucp/Poll** file specifies when the Basic Networking Utilities (BNU) program should poll (initiate automatic calls to) designated remote computers. This file is used in conjunction with the **/usr/spool/cron/crontabs/uucp** file, the **uudemon.hour** command, and the **uudemon.poll** command. Together, these files are responsible for initiating automatic calls to certain remote systems. Modify the times specified in the **Poll** file based on how the systems at your site are used.

Each entry in the **Poll** file contains the name of the remote computer followed by a sequence of times when the BNU program should poll that system. Specify times as digits between 0 and 23. The format of the entry is as follows:

SystemName Time [*Time* ...]

The fields in the **Poll** file entry must be separated by at least one space. More spaces can be used for readability. A tab character between the *SystemName* field and the first *Time* field is optional.

Note: Only someone with root user authority can edit the **Poll** file, which is owned by the **uucp** program login ID.

Example

Following is a standard entry in the **Poll** file:

```
hera <TAB> 0 4 8 12 16 20
```

This entry specifies that the local system will poll the remote system `hera` every 4 hours.

The tab character can be replaced by one or more spaces. Thus the preceding entry is equivalent to the following one:

```
hera    0 4 8 12 16 20
```

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Most versions of UUCP require a tab character between the *SystemName* field and the first *Time* field. In AIX BNU, either a tab or spaces will work.

Files

/etc/locks directory	Contains lock files that prevent multiple uses of devices and multiple calls to systems.
/usr/spool/cron/crontabs/uucp file	Schedules BNU jobs for the **cron** daemon.

Related Information

Configuring BNU and Understanding the BNU Configuration Files in *Communication Concepts and Procedures*.

The **cron** daemon runs the commands that poll remote systems.

The **uucpadm** command can be used to make entries in the **Poll** file.

The **uudemon.hour** command and **uudemon.poll** command poll remote systems.

How to Set Up BNU Polling of Remote Systems in *Communication Concepts and Procedures*.

BNU Systems File Format

Purpose

Lists remote computers with which users of the local system can communicate using the Basic Networking Utilities (BNU) program.

Description

The **/usr/lib/uucp/Systems** file lists the remote computers with which users of the local system can communicate using the Basic Networking Utilities (BNU) program. Each entry in the **Systems** file represents a remote system, and users on the local system cannot communicate with a remote system unless that system is listed in the local **Systems** file. A **Systems** file must be present on every computer at your site that uses the BNU facility.

Each entry in the **Systems** file contains the following items:

- The name of the remote system
- The times when users can connect to the remote system
- The type of link (direct line or modem link)
- The speed of transmission over the link
- The information needed to log in to the remote system.

Note: When a remote system not listed in the **Systems** file attempts to contact the remote system, the BNU program calls the **/usr/lib/uucp/remote.unknown** shell procedure.

Only someone with root user authority can edit the **Systems** file, which is owned by the **uucp** program login ID.

Fields in the Systems File

Each entry in the **Systems** file is a logical line containing fields and optional subfields. These fields appear in the following order:

SystemName Time[;*RetryTime*] *Type*[,*ConversationProtocol*] *Class Phone Login*

SystemName Lists the name of the remote system.

Time Lists the times when the local system can contact the remote system.

RetryTime Specifies the minimum time after which the BNU program will attempt to contact the remote system again, when the first contact was unsuccessful. This field is optional.

Type Defines the type of connection used to contact the remote system.

ConversationProtocol
Defines the conversation protocol used over a TCP/IP connection. Use only when the *Type* field entry is `TCP`.

Class Defines the speed of transmission over the connection.

Phone Specifies the phone number of the modem at the remote system.

Login Defines the login sequence, or chat script, for the remote system.

There must be an entry in every field of a line in the **Systems** file. If a field does not apply to the particular remote system (for example, a hardwired connection would not need a telephone number in the *Phone* field), use a – (hyphen) as a placeholder.

Lines in the **Systems** file cannot wrap. Each entry must be on only one line in the file. However, the **Systems** file can contain blank lines and comment lines. Comment lines begin with a # (pound sign). Blank lines are ignored.

SystemName Field

The *SystemName* field contains the name of the remote system. You can list an individual remote system in the **Systems** file more than once. Each additional entry for a system represents an alternate communication path that the BNU program uses in sequential order when trying to establish a connection between the local and the remote system.

Time Field

The *Time* field contains a string that indicates the days of the week and the times of day during which users on the local system can communicate with the specified remote system. For example, the `MoTuTh0800-1730` string indicates that local users can contact the specified remote system on Mondays, Tuesdays, and Thursdays from 8 a.m. until 5:30 p.m.

The day part of the entry can be a list including any day or days represented by `Mo`, `Tu`, `We`, `Th`, `Fr`, `Sa`, or `Su`. The day entry may also be `Wk` if users can contact the remote system on any weekday, or `Any` if they can use the remote system on any day of the week including Saturday and Sunday.

Enter the time at which users can contact the remote system as a range of times, using the 24-hour clock notation. For example, if users can communicate with the specified remote system only during the morning hours, type a range such as `0800-1200`. If users can contact the remote computer at any time of day or night, simply leave the time range blank.

It is also possible to specify times during which users cannot communicate with the remote system by specifying a time range that spans `0000`. For example, typing `0800-0600` means that users can contact the specified system at any time *except* between 6 a.m and 8 a.m. This is useful if a free line is needed at a certain time of day in order to use the remote system for administrative purposes.

If the remote system calls the local system, but users on the local system cannot call the remote system, the time entry may be `Never`.

To include multiple *Time* fields, separate them with a `,` (comma). For example, `Wk1800-0600,Sa,Su` means that users can contact the remote system on any weekday at any time except between the hours of 6 p.m. and 6 a.m., and at any time on Saturday and Sunday.

RetryTime Subfield

The *RetryTime* subfield is an optional subfield that specifies the minimum time in minutes between an unsuccessful attempt to reach the remote system and the retry time when the BNU program again attempts to communicate with that system. This subfield is separated from the rest of the string by a `;` (semicolon). For example, `Wk1800-0600,Sa,Su;2` indicates that if the first attempt to establish communications fails, BNU should continue to attempt to contact the remote system at no less than 2-minute intervals.

Note: This subfield, when present, overrides the default retry time of 5 minutes.

The retry time does *not* cause BNU to attempt contact with the system once the time has elapsed. It specifies the *minimum* time BNU must wait before attempting to contact the remote system.

Type Field

The *Type* field identified the type of connection used to communicate with the remote system. The available types of connections are `ACU` for a telephone connection using a modem, the remote system name (as in the *SystemName* field) for a hardwired connection, and `TCP` for a connection using TCP/IP. There must be a corresponding entry for the type of connection in the **/usr/lib/uucp/Devices** file.

ConversationProtocol Subfield

If you use the `TCP` entry in the *Type* field, the *ConversationProtocol* subfield, associated with the caller, specifies a conversation protocol. The default is the **g** protocol. To use a different subfield, enter it with a , (comma) and the letter representing one of the other conversation protocols, either the **t** or **e** protocol. These protocols are faster and more efficient than the **g** protocol.

Protocol	Explanation
g	This is the default. The **g** protocol is preferred for modem connections, but it involves a large overhead in running BNU commands because it uses the checksumming and packetizing functions.
t	The **t** protocol presumes an error-free channel, and is essentially the **g** protocol without the checksumming and packetizing functions. Use the **t** protocol: • To communicate with a site running the AIX version of the BNU program • To communicate with a site running the Berkeley version of the UNIX-to-UNIX Copy Program (UUCP). The **t** protocol is not reliable for modem connections.
e	Use the **e** protocol: • To communicate with a site running the AIX version of the BNU program • To communicate with a site running a non-AIX version of the BNU program. The **e** protocol is not reliable for modem connections.

Use either the **t** or **e** protocol to communicate with a site running the AIX version of the BNU program. Use the **e** protocol for a site running a non-AIX version of the BNU program. Use the **t** protocol for sites running the Berkeley version of the UNIX-to-UNIX Copy Program (UUCP).

Class Field

The *Class* field typically specifies the speed at which the specified hardwired or telephone line transmits data. It is generally `300, 1200, 2400` or higher for a hardwired device, and `300, 1200,` or `2400` for a telephone connection.

This field can also contain a letter with a speed (for example, `C1200, D1200`) to differentiate between classes of dialers. For example, some offices have more than one telephone network, one for internal use and one for external communications. In such a case, it is necessary to distinguish which lines should be used for each connection.

If the entry in the *Type* field is `ACU`, the *Class* field in the **Systems** file is matched against the *Class* field in the **/usr/lib/uucp/Devices** file to find the device to use for connections. For example, if the **Systems** file entry for system `hera` is:

```
hera Any ACU 1200 3–3–5–2 ogin: nuucp ssword: oldoaktree
```

BNU searches for an entry in the **Devices** file with a *Type* of `ACU` and a *Class* of `1200` and connects to system `hera` using the first available device that meets these specifications.

If the device can match any speed, enter the word `Any` in the *Class* field. Note that the word `Any` begins with an uppercase `A`.

Do not include a transmission rate for a TCP/IP connection. If you do not type a transmission rate in the *Class* field, use a – (hyphen) as a placeholder.

Phone Field

For a telephone connection over a modem, the *Phone* field specifies the telephone number used to reach the remote modem. If this entry represents a hardwired connection, type a hyphen as a placeholder. If this entry represents a telephone connection using a modem, type the remote modem's phone number.

The *Phone* field for a telephone connection must include all of the following items that apply, in the following order:

- The code for an outside line
- Any long-distance access codes
- A 1 (one) plus the area code (if the modem is out of the local area)
- The three-digit exchange number
- The four-digit modem number.

Entering the entire phone number is the most efficient method of including phone numbers if your site uses only a relatively small number of telephone connections. However, if your site includes a large number of remote connections established using a phone line and a modem, you may prefer to use the **/usr/lib/uucp/Dialcodes** file to set up dial-code abbreviations.

For example, if your site communicates regularly using modems to many other systems all located at the same remote site, it is more efficient to use a dial-code abbreviation in the **Systems** file than to type the complete phone number of each remote modem.

The dial-code entry in the **/usr/lib/uucp/Dialcodes** file defines an alphabetic abbreviation that represents the following portions of the phone number:

- The code for an outside line
- Any long-distance access codes
- A 1 (one) plus the area code (if the modem is out of the local area)
- The three-digit exchange number.

In the *Phone* field in the **Systems** file entry, type the alphabetic abbreviation followed by the four-digit modem number.

Note: Enter the alphabetic abbreviation in the **/usr/lib/uucp/Dialcodes** file *only once* for all the remote modems listed in the **Systems** file. Then use the same abbreviation for all entries in the **Systems** file for modems at that site.

For callers that are actually switches, the *Phone* field is the token the switch requires to get to the particular computer. The token you enter here is used by the functions specified in the *Type* field of the **/usr/lib/uucp/Devices** file.

Login Field

The *Login* field specifies login information that the remote system must receive before allowing the calling local system to establish a connection. The *Login* field is a series of fields and subfields called *expect-send* characters.

Expect-Send Characters in *Login* Fields

Enter the required login information in the following form:

[Expect Send] ...

The *Expect* subfield contains characters that the local system expects to receive from the remote system. Once the local system receives those characters, it sends another string of characters that comprise the *Send* subfield.

For example, the first *Expect* subfield generally contains the remote system's login prompt, and the first *Send* subfield generally contains the login ID of the remote system. The second *Expect* subfield contains the remote password prompt, and the second *Send* subfield contains the remote system password.

The *Expect* subfield may include subfields entered in the following form:

Expect[*-Send-Expect*] ...

In this case, the first *Expect* subfield still represents the string that the local system expects to receive from the remote system. However, if the local system does not receive (or cannot read) that first *Expect* string, it sends its own string (the *Send* string within the brackets) to the remote system. The local system then expects to receive another *Expect* string from the remote system.

For example, the *Expect* string may contain the following characters:

```
login:—login:
```

The local system expects to receive the `login:` string. If the remote system sends that string and the local system receives it correctly, the BNU program goes on to the next field in the expect-send sequence. However, if the local system does not receive the `login:` string, it sends a null character followed by a new line, and then expects to receive a second `login:` string from the remote computer.

If the remote system does not send an *Expect* string to the local system, type "" (two double quotation marks), representing a null string, in the first *Expect* subfield.

Every time the local system sends a field, it automatically transmits a new line following that *Send* subfield. To disable this automatic new line, type `\c` (a backslash and the letter c) as the last two characters in the *Send* string.

Two special strings can be included in the login sequence. The `EOT` string sends an ASCII EOT (end of transmission) character, and the `BREAK` string attempts to send an ASCII BREAK character.

Valid Expect-Send Sequences

Following are the valid expect-send strings for the *Login* field:

String	Explanation
\N	Null character.
\b	Backspace character.
\c	At the end of a field, suppress the new line that normally follows the characters in a *Send* subfield. Otherwise, ignore this string.
\d	Delay 2 seconds before sending or reading more characters.
\p	Pause for approximately 1/4 to 1/2 second.
\E	Turn on the echo check.
\e	Turn off the echo check.
\K	Send a BREAK character. This is the same as entering BREAK. This character can be used to cycle a modem's speed.
\n	New-line character.
\r	Carriage return.
\s	Space character.
\t	Tab character.
\\	Backslash character.
EOT	EOT character. When you enter this string, the system sends two EOT new-line characters.
BREAK	BREAK character. This character can be used to cycle a modem's speed.
\ddd	Collapse the octal digits (ddd) into a single character and send that character.

Using the BREAK Character to Cycle a Modem

A BREAK or \K character is usually sent to cycle the line speed on computers that have a multi-speed modem. For example, if you are using a 2400 baud modem to contact a remote system with a multi-speed modem that normally answers the phone at 9600 baud, you can begin the chat script for that system with a \K character to cause the remote system's modem to cycle down to 2400 baud.

Entries for Use with TCP/IP

If your site is using TCP/IP, include the relevant TCP/IP entries in the **Systems** file. For a remote system connected to the local system using TCP/IP, the entries in the *SystemName*, *Time,* and *Login* fields are the same as for a remote system using any other type of connection. For the *Type* field, decide on the appropriate TCP/IP conversation protocol to enter in the TCP *ConversationProtocol* subfield. Enter TCP followed by a , (comma) followed by the letter representing the protocol. In the *Class* and *Phone* fields, enter a – (hyphen) as a placeholder.

Examples

Setting Up Entries Using Modems

1. A standard entry for a telephone connection using a modem looks like this:

   ```
   merlin 0830-1730 ACU 1200 123-4567 in:—in: uucp1 word: rainday
   ```

   This entry allows users to contact system `merlin` daily between 8:30 a.m. and 5:30 p.m. using an `ACU` at `1200` bps. The telephone number is `123-4567`. The login name on `merlin` is `uucp1` and the password is `rainday`. The local system expects the phrase `in:` before it sends the login name. If the local system does not receive the phrase `in:`, it sends a null character and a new-line character and expects the phrase again.

2. To use a `1200` baud modem to contact a system with a multi-speed modem, make an entry similar to the following:

   ```
   athena Any ACU 1200 123-7654 \K\K in:—in: uucpa word: shield
   ```

   The `\K` prefacing the login script instructs the remote modem to cycle down one speed. If the modem has three speeds, `9600`, `2400`, and `1200`, the first `\K` character causes it to cycle to the `2400` baud setting, and the second `\K` character causes it to use the `1200` baud setting. (A third `\K` causes the modem to start the cycle over by returning to `9600` baud.)

Setting Up Entries Using Direct Connections

3. A standard entry for a hardwired connection between a local and a remote system looks like this:

   ```
   hera  Any  hera  1200  -  login:—login: uzeus  word: thunder
   ```

   The remote system is `hera`, which can be called at any time. The entry in the *Type* field is also `hera`, indicating a directory connection at `1200` bps (the *Class* field). There is a placeholder in the *Phone* field since no telephone number is necessary.

Setting Up Entries Using TCP/IP Connections

4. In order to make the appropriate entries in the **Systems** file, decide on the appropriate TCP/IP conversation protocol to enter in the TCP *Caller* subfield. For example, Enter the following in the **Systems** file to use TCP/IP to connect to system `venus` with the default **g** protocol:

   ```
   venus  Any  TCP  -  -  in:—in: uzeus  word: lamplight
   ```

   Replace the *send* and *expect* characters in the sample login field with the login prompt, login, password prompt, and password appropriate to the remote system for which you are establishing a connection.

Using Dialcode Abbreviations

5. To use a dialcode abbreviation defined in the **/usr/lib/uucp/Dialcodes** file, enter the following in the **Systems** file:

   ```
   merlin Any ACU 1200 local8784 in:—in: uucp1 word: magic
   ```

   This assumes that an entry for the dial code `local` exists in the **Dialcodes** file. For example, the following entry

   ```
   local 9=445
   ```

   in the **Dialcodes** file would cause BNU to expand the telephone number as `9=4458784`.

Setting Up Entries for Both Local and Remote Systems

6. For a direct connection between two systems, the **Systems** file on system `zeus` contains the following entry for the remote system `hera`:

   ```
   hera Any hera 1200 - "" \r\d\r\d\r in:—in: uzeus word: thunder
   ```

   The **Systems** file on system `hera` contains the following entry for system `zeus`:

   ```
   zeus Any zeus 1200 - "" \r\d\r\d\r in:—in: uhera word: lostleaf
   ```

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/usr/lib/uucp directory	Contains all the configuration files for BNU, including the **Systems** file.
/usr/lib/uucp/Devices file	Contains information about available devices.
/usr/lib/uucp/Dialcodes file	Contains dialing code abbreviations.
/usr/lib/uucp/Permissions file	Describes access permissions for remote systems.
/usr/lib/uucp/remote.unknown file	Records contacts from unknown systems.

Related Information

Understanding the BNU Configuration Files and Configuring BNU in *Communication Concepts and Procedures.*

How to Configure BNU, How to Configure BNU for Use with TCP/IP, How to Monitor a BNU Remote Connection, and How to Use the **uucico** Daemon to Debug BNU Login Failures in *Communication Concepts and Procedures.*

The **uucpadm** command can be used to make entries in the **Systems** file.

The **uucico** daemon, **uusched** daemon, **uucpd** daemon, **uuxqt** daemon.

The **mail** command, **sendmail** command, **uucp** command, **uuname** command, **uuto** command, **uutry** command, **Uutry** command, **uukick** command, **uux** command.

EM78 Customization File Format

Purpose

Specifies the keyboard layout, screen colors, and field attribute modes to be used in 3278/79 Terminal Emulation.

Description

An EM78 Customization file contains the settings used by the EM78 Emulation program. These settings affect the following aspects of the emulation:

- Key Definitions
- Screen Colors
- Field Attribute Colors and Modes.

EM78 is delivered with two customization files, the **/usr/lib/em78/emdefs.p** file and the **/usr/lib/em78/emaltdefs.p** file. These files can be used as delivered. You can also copy one of the files to customize to change the key, color, or field attribute settings, or you can create a new customization file. The selected customization file is then installed using the **emkey** command.

The edited customization file can be stored in any directory and have any name. The user must specify the path name and name of the file to the **emkey** command using the **–i** flag.

Any text to the right of the required parts of a line in the customization file is interpreted as a comment. Should the comment overflow the line, the next line must begin with an * (asterisk).

Any line beginning with an asterisk is interpreted as a comment.

Warning: In the **emdefs.p** and **emaltdefs.p** file, the **quit** function is assigned to the Ctrl + D workstation key combination. If you reassign the Ctrl + D key combination to a different function, you must assign the quit function to another key combination. Otherwise, you will not be able to leave the emulator.

Note: It is recommended that you edit a *copy* of the **emdefs.p** or **emaltdefs.p** file and keep the originals intact in case you want to return the emulator settings to the default settings.

Key Definitions

The customization file maps keys on the local workstation to keys on the 3278/79 keyboard. The file contains a list of key numbers and mnemonic key names that refer to local workstation keys, mapped to key numbers that refer to 3278/79 keys.

Key Number Mappings

The format of a key definition entry is as follows:

WorkstationKeyName *3278/79KeyName* | **none** [*Comments*]

The fields in the key definition entries must be separated by at least one space. More spaces may be used to make the entry easier to read.

The fields in the key definition entry are:

WorkstationKeyName	Specifies a valid workstation key name to be mapped to a 3278/79 key or emulator function.
3278/79KeyName	Specifies a valid 3278/79 key name or one of the EM78 functions such as **asciitoggle**.
none	Removes the assignment for a key.

Warning: You must assign the **quit** function to a workstation key combination in the configuration file. Otherwise, you will not be able to leave the emulator.

Mnemonic Key Mappings

Some keys on the workstation are defined with mnemonic key names. These keys must usually be mapped to a combination of 3278/79 keys. The format of a mnemonic key definition entry is:

MnemonicKeyName 3278/79KeyName1 + *3278/79KeyName2* [*Comment*]

where *MnemonicKeyName* is a valid workstation mnemonic key name, and *3278/79KeyName1* and *3278/79KeyName2* are valid 3278/79 key names.

The fields in the Mnemonic Key Mapping entries must be separated by at least one space. The + (plus sign) must have at least one space preceding and at least one following. More spaces may be used to make the entries easier to read.

Screen Colors

Screen colors are defined in the customization file by defining four color combinations and then specifying which of the four color combinations is to be used for the default color and status line color.

Color Definitions

The format of the color definition entries is as follows:

ColorName Background + *Foreground* [*Comments*]

OR

ColorName Foreground + *Background* [*Comments*]

The fields in the Screen Colors entries must be separated by at least one space. The + (plus sign) must have at least one space preceding and at least one following. More spaces may be used to make the entries easier to read.

The fields in the color definition lines are:

ColorName	Specifies the color combination being defined. Up to four color combinations, **COLOR1**, **COLOR2**, **COLOR3**, and **COLOR4**, may be defined.
Background	Names the background color, indicated by using `b_` followed by the color, as in `b_black`.
Foreground	Names the foreground color, indicated by using `f_` followed by the color, as in `f_ltgreen`.

Default Color

The format of the default color entry is as follows:

default *ColorName* [*Comments*]

where the *ColorName* variable specifies any of the previously defined color combinations, **COLOR1**, **COLOR2**, **COLOR3**, or **COLOR4**.

Status Color

The status color line defines the colors used on the status line, also called the operator information area. The format of the status color entry is as follows:

status *ColorName* | **default** [*Comments*]

The status color may be defined, using the *ColorName* variable, as any of the previously defined color combinations, or it may be defined as **default**.

Note: The default color may *not* be defined as **status**.

Field Attribute Colors and Modes

The field attributes section assigns colors and modes to 3278/79 field attribute combinations. The format of an entry in the field attributes section is as follows:

FieldAttributeName *ColorName* | **default** [**+** *Mode*] [*Comments*]

The fields in the *Field Attribute* entries must be separated by at least one space. The **+** (plus sign) must have at least one space preceding and at least one following. More spaces may be used to make the entries easier to read.

The fields in the Field Attribute definition lines are:

FieldAttributeName Specifies one of the recognized 3278/79 field attribute codes.

ColorName Specifies one of the previously defined color combinations (for example, **COLOR1**) or the word **default**. The color or default color used must be defined in the **Screen Colors** section of the customization file.

Mode Specifies one of the four recognized modes. They are:

blink Causes the field to blink.

bold Causes the field to appear in bold type.

underline Causes the contents of the field to be underlined.

revideo Causes the field to appear in reverse video.

The *Mode* entry is optional.

Note: Whether your system supports all four modes depends on the hardware. If you specify a mode that your display does not support, no change will take place.

Examples

Defining Keys

1. To map a workstation key to a 3278/79 key, edit the configuration file and enter a line similar to the following:

   ```
   k126_c  k32_1  attn (attention)
   ```

   This line maps the workstation Ctrl + Pause key combination to the 3278/79 Attn (attention) key.

2. To map a workstation key to an EM78 function, enter:

   ```
   k125_c  screenrepl
   ```

   This line maps the workstation Ctrl + Scroll Lock key combination to the emulator **screenrepl** function.

3. To reassign a key using the **none** statement, edit the configuration file and enter a line similar to the following:

   ```
   k48_c shell

   k48_c none  /*Remove the previous definition shell.*/

   k31_c shell /* The control state of key 31 is shell.*/
   ```

   These lines assign the **shell** function to the workstation key `k31_c` instead of to key `k48_c`.

4. To define a mnemonic key, edit the configuration file and enter a line similar to the following:

   ```
   e_umlaut  k32_9_s + k32_24
   ```

   This line assigns the `e_umlaut` character to the 3278/79 key combination of `k32_9_s` followed by `k32_24`.

Defining Screen Colors

1. To define the color combinations, enter combinations similar to the following:

   ```
   COLOR1    f_cyan + b_black
   COLOR2    f_ltgreen + b_black
   COLOR3    f_red + b_black
   COLOR4    f_white + b_black
   ```

   The first line defines color combination one (`COLOR1`) as a foreground of `cyan` against a background of `black`. Similarly, using color combination three (`COLOR3`) will cause a foreground of `red` to be displayed against a `black` background.

2. To define the default color edit the configuration file and enter a line similar to the following:

   ```
   default  COLOR2
   ```

   If the colors are defined as in example 1, this line will cause the screen to be displayed in `light green` against a `black` background.

3. To define the status color as the default color, edit the configuration file and enter a line similar to the following:

   ```
   status  default
   ```

   If the default color is defined as in examples 1 and 2, this line will cause the status line to be displayed in `light green` against a `black` background.

4. To define the status color as a previously defined color combination, edit the configuration file and enter a line similar to the following:

   ```
   status  COLOR3
   ```

   If the colors are defined as in example 1, this line will cause the status line to be displayed in `red` against a `black` background.

Defining Field Attribute Colors and Modes

1. To define an attribute color and a mode, edit the configuration file and enter a line similar to the following:

   ```
   uahd  COLOR3 + bold
   ```

   This line causes any field which is unprotected, alphanumeric, intensified display, and detectable with a light pen (`uahd`) to be displayed in bold type using the previously defined `COLOR3` combination.

2. To define a color without a mode, edit the configuration file and enter a line similar to the following:

   ```
   unlx  COLOR2
   ```

   This line causes any field which is unprotected, numeric, normal intensity, and non-detectable with a light pen (`unlx`) to be displayed using the previously defined `COLOR2` color combination.

3. To define an attribute color as the default color, edit the configuration file and enter a line similar to the following:

   ```
   uaix  default
   ```

   This line causes any field which is unprotected, alphanumeric, non-display, and non-detectable with a light pen (`uaix`) to be displayed using the previously defined `default` color combination.

Implementation Specifics

This file is part of 3278/79 Emulation in AIX 3278/79 Emulation/6000.

Files

/usr/lib/em78/emdefs.p	Contains default customization settings for EM78.
/usr/lib/em78/emaltdefs.p	Contains alternate customization settings for EM78.

Related Information

The **emkey** command, **em78** command.

EM78 Emulator Colors, EM78 Field Attribute Codes, EM78 Functions, How to Customize EM 78, and Mapping Keys for EM78 in *Communications Concepts and Procedures*.

HCON e789_ctbl.p File Format

Purpose

Contains the source for the HCON default color definition table.

Description

The **/usr/lib/hcon/e789_ctbl.p** file contains the source for the AIX 3270 Host Connection Program/6000 (HCON) default color definition table (the **/usr/lib/hcon/e789_ctbl** file). A copy of the **e789_ctbl.p** file can be modified and used as input to the **e789cdef** command in order to customize the color definition table.

The **e789_ctbl.p** file specifies the colors, attributes, and highlights to be displayed for each 3270 field on the screen. HCON supports the 3270 extended data stream, seven host colors, and extended highlighting. The 3270 data stream may contain color and highlighting information supplied by a host application.

The colors, attributes, and highlights displayed on the terminal are based on the value of the host color and highlighting. The foreground and background colors and the highlighting to be used on the terminal must be specified in the color definition file. The colors and highlighting supported depend on the type of display in use.

The HCON color utility (the **e789cdef** command) uses color definition lines to map the color and highlighting attributes of the 3270 terminals to the characteristics of the terminal displays. The **e789_ctbl.p** file contains the color definition lines.

The general format of a color definition line is as follows:

3270Display = *Foreground Background Highlight*

Each line consists of a host color or attribute followed by a host highlight, followed successively by a terminal foreground color name, a background color name, and a highlight name. Leading and trailing blanks on the definition line are ignored. However, the names must be separated by at least one space or tab character.

Everything that follows an * (asterisk) on a line is considered a comment.

The *3270Display* portion of the color definition line may be any of the following:

3270Color 3270Highlight
: Indicates 3270 color and highlight information supplied by the host application, and is referred to as *seven color mode*.

3270Attribute 3270Highlight
: Indicates 3270 attribute and highlight information supplied by host applications that do not use 3270 colors, and is referred to as *four color mode*.

status
: Defines the status line (Operator Information Area).

unfrmt
: Defines an *unformatted screen*, where the host application does not supply any field attributes or color information.

Following are the possible values for the variables that can be used in the *3270Display* portion of the color definition line:

3270Color The name of one of the colors provided on 3270 displays, which are:

blue
red
green
yellow
turquoise
pink
white
defnorm No color specified; normal brightness.
defint No color specified; intensified brightness.

Note: The names of these colors are defined in the **/usr/lib/hcon/nls_names** file and can be changed.

3270Highlight One of the extended highlighting characteristics provided on 3270 displays, which are:

none
underline
blink
reverse

Note: The names used to designate extended highlighting characteristics are defined in the **/usr/lib/hcon/nls_names** file and can be changed.

3270Attribute One of the following keywords, specifying certain characteristics of 3270 fields:

unorm Unprotected normal
uint Unprotected intense
pnorm Protected normal
pint Protected intense.

Note: The names used to designate field characteristics are defined in the **/usr/lib/hcon/nls_names** file and can be changed.

Following are the possible values for the variables that define the terminal display:

Foreground The name of the color to be used for the foreground

Background The name of the color to be used for the background.

The names of the terminal colors are:

black
red
green
yellow
blue
magenta
cyan
white

Note: The color names used for the *Foreground* and *Background* colors are defined in the **/usr/lib/hcon/nls_names** file and can be changed.

Even if a color definition table is used for printer-emulation sessions, the background color is always white.

Highlight One of the following terminal highlighting attributes:

none	
bright	Turn brightness on, regardless of field attribute.
reverse	Use reverse video, regardless of 3270 extended highlighting attribute.
blink	Blink field.
underline	Underline field.

Note: The highlighting available depends on the type of highlighting supported on your terminal.

The names used to designate the types of terminal highlighting are defined the **/usr/lib/hcon/nls_names** file and can be changed.

Examples

1. To map a field by its host attributes, use a line similar to the following:

   ```
   unorm underline = green black underline
   ```

   This color definition line will cause an unprotected, normal field (`unorm` attribute) underlined on the host computer to be displayed on the emulating terminal in green on a black background, and underlined.

2. To map a field by its host color, use a line similar to the following:

   ```
   pink blink = magenta black bright
   ```

   This color definition line causes a field that is pink and blinking on the host computer to be displayed on the emulating terminal in magenta on a black background, with bright highlighting.

3. To define the display of the status line, use a line similar to the following:

   ```
   status = green black none
   ```

   This color definition line causes the status line (`status`) on the emulating terminal to be displayed in green on a black background, without highlighting (`none`).

4. To define the display of an unformatted screen, use a line similar to the following:

   ```
   unfrmt = green black none
   ```

   This color definition line causes an unformatted host screen (`unfrmt`) to be displayed on the emulating terminal in green on a black background, without highlighting (`none`).

Implementation Specifics

This file is part of the AIX 3270 Host Connection Program/6000 (HCON).

Files

/usr/lib/hcon directory	Contains HCON files, including the **e789_ctbl.p** file.
/usr/lib/hcon/e789_ctbl file	Contains the default binary color definition table.
/usr/lib/hcon/nls_names file	Contains color, highlighting, and attribute names used in color definitions.

Related Information

The **e789cdef** command creates a binary color definition table based on color definition lines.

How to Customize the HCON Color Definition Table in *Communication Concepts and Procedures* discusses creating customized tables.

To start SMIT for use with HCON, use the **smit hcon** command.

To set up a session profile for HCON, use SMIT or the **mkhcons** command. To modify an existing profile, use SMIT or the **chhcons** command.

Customizing HCON in *Communication Concepts and Procedures* discusses the options available to you for customizing HCON colors, keyboard definitions, and key and color names.

HCON e789_ktbl.p File Format

Purpose

Contains the source for the HCON default keyboard definition table.

Description

The **/usr/lib/hcon/e789_ktbl.p** file contains the default keyboard mappings source for the AIX 3270 Host Connection Program/6000 (HCON). The source mappings in the **e789_ktbl.p** file correspond to those in the binary **/usr/lib/hcon/e789_ktbl** file. A copy of the **e789_ktbl.p** file can be modified and used as input to the **e789kdef** command in order to customize the keyboard definition table.

The **e789_ktbl.p** file consists of a series of key definition lines in the following format:

KeyName *HCONFunctionName* [**Comments*]

The *KeyName* must be separated from the *HCONFunctionName* by at least one space. More spaces can be used to increase readability. Everything that follows an * (asterisk) on a line is considered a comment. Comments are optional.

KeyName — Corresponds to a key sequence on an HCON-supported keyboard.The possible values for the *KeyName* variable are defined in the **/usr/lib/hcon/keynames** file.

HCONFunctionName — Corresponds to a character or function supported by the HCON emulator. The valid values for the *HCONFunctionName* variable are defined in the **/usr/lib/hcon/func_names** file.

Examples

1. To map alphanumeric symbols to key sequences, create lines similar to the following:
   a. Map the ¿ (inverted question mark) to the @ (at sign) key on the HCON supported keyboard with the following entry:

      ```
      @      inverted_question
      ```

      When the user presses the @ key, an inverted question mark appears.

   b. Map the £ (pound sterling sign) to the $ (dollar sign) key on the HCON supported keyboard with the following entry:

      ```
      $      english_pound
      ```

      When the user presses the $ key, a pound sterling sign appears.

2. To map HCON functions to key sequences, create lines similar to the following:
   a. Map the DELWORD emulator function to the Ctrl-W key sequence on any HCON supported keyboard using the following entry:

      ```
      cntrl_w DELWORD
      ```

   b. Map the REFRESH function to the Ctrl-G key sequence using the following entry:

      ```
      cntrl_g REFRESH
      ```

c. Map the NEXT function to the key sequence corresponding to the key_pa2 key name using the following entry:

```
key_pa2 NEXT
```

d. Map the ENTER function to the key sequence corresponding to the key_newline key name using the following:

```
key_newline ENTER
```

Implementation Specifics

This file is part of the AIX 3270 Host Connection Program/6000 (HCON).

Files

/usr/lib/hcon directory	Contains HCON files, including the **e789_ktbl.p** file.
/usr/lib/hcon/e789_ktbl file	Contains the default binary keyboard definition table.
/usr/lib/hcon/func_names file	Contains HCON function names used in keyboard definition lines.
/usr/lib/hcon/keynames file	Contains HCON keynames used in keyboard definition lines.

Related Information

Use the **e789kdef** command creates a binary keyboard definition table based on keyboard definition lines.

How to Customize the HCON Keyboard Definition Table in *Communication Concepts and Procedures* discusses creating a customized keyboard table.

To start SMIT for use with HCON, use the **smit hcon** command.

To set up a session profile for HCON, use SMIT or the **mkhcons** command. To modify an existing profile, use SMIT or the **chhcons** command.

HCON func_names File Format

Purpose

Contains function names for HCON.

Description

The **/usr/lib/hcon/func_names** file contains the function names for the AIX 3270 Host Connection Program/6000 (HCON). These names are used as input to the **e789kdef** (keyboard redefinition) command.

The **func_names** file allows function names to be translated into their equivalent in any language. Thus the character string that signifies a function name may change, but the internal identifier HCON uses for the function name remains constant.

The **func_names** file lists emulator special keys that can be used as input when redefining a keyboard with the HCON **e789kdef** command. Emulator special keys include functions that appear on IBM 3278/79 keyboards (such as PF1 and ENTER) and functions special to the HCON emulator program (such as DELWORD and FTABW) that move the cursor, add and delete text, and switch active emulator screens.

Entries in the **func_names** file are in the following format:

Name=*InternalID* [* *Comments*]

The mapping string *Name*=*InternalID* must not contain any spaces.

The *Name* variable can be translated into any equivalent form of the function or key name. However, the length of the *Name* variable cannot be more than 36 characters.

The *InternalID* variable must not be changed.

Any line that starts with an * (asterisk) and any part of a line following an * are interpreted as comments by the **e789kdef** command.

Note: Changes to names in the **func_names** file affect the entire keyboard customization process. All users must use the new names as input to the **e789kdef** command.

Examples

Following are sample entries from the **func_names** file:

1. Example of a function key mapping:

```
NEWLINE=0x09 * the function "newline" has an id of 9
```

2. Example of a non-U.S. English key mapping:

```
e_acute=0x82
```

3. Example of a mapping for the no-function function:

```
NOFUNC=0x400 *the function "nofunction" has an id of 1024
```

Implementation Specifics

This file is part of the AIX 3270 Host Connection Program/6000 (HCON).

Files

/usr/lib/hcon directory	Contains HCON files, including the **func_names** file.
/usr/lib/hcon/e789_ktbl file	Contains the default binary keyboard definition table.
/usr/lib/hcon/e789_ktbl.p file	Contains the source for the default binary keyboard definition table.
/usr/lib/hcon/keynames file	Contains HCON keynames.

Related Information

Names defined in the **func_names** file are used as input to the **e789kdef** command and also to customize the **e789_ktbl.p** file.

Customizing HCON in *Communication Concepts and Procedures* discusses the options available to you for customizing HCON keyboard definitions as well as function and key names.

HCON keynames File Format

Purpose

Identifies HCON key names.

Description

The **/usr/lib/hcon/keynames** file lists all of the AIX 3270 Host Connection Program/6000 (HCON) defined key names. In HCON, a key name is an identifier that corresponds to a non-U.S. English character or to specific keys on keyboards supported by HCON. Use the key names listed in the **keynames** file when redefining keys with the **e789kdef** command.

The **keynames** file allows key names to be translated into their equivalent in any language. Thus the character string that signifies a key name may change, but the internal identifier HCON uses for the key remains constant.

Key name mappings are in the following format:

Name=InternalID [* *Comments*]

The key mapping *Name=InternalID* string must not contain any spaces.

The *Name* variable can be translated into its equivalent in any language. However, the length of the *Name* variable cannot be more than 36 characters.

The *InternalID* variable must not be changed.

Any line beginning with an * (asterisk) and any part of a line following an * are interpreted as comments by the **e789kdef** command.

Note: Changes to key names in the **keynames** file affect the entire keyboard customization process. All users must use the new names as input to the **e789kdef** command.

Key names may be either alphanumeric or non-alphanumeric. Alphanumeric names describe a letter, number, symbol, or a non-U.S. English character.

Some alphanumeric keys have special functions when pressed in combination with the Ctrl key. This type of key name is denoted by **cntrl_***n*, where *n* is any lowercase alphabetic character (a through z).

Nonalphanumeric, noncontrol-key names (such as **key_home**, **key_pf1**, and **key_left**) are prefixed with the characters **key_**. These key names are mapped to emulator functions in the default **e789_ktbl.p** file.

Due to NLS support requirements and the different placement and number of keys on the keyboards supported by HCON, some key names must be mapped to different physical keys. However, the **e789kdef** command is not affected by the physical mapping.

Examples

Following are sample entries from the **keynames** file:

1. Examples of control key mappings:

```
cntrl_a=0x01
cntrl_b=0x02
```

2. Examples of symbol key mappings:

```
%=0x25
&=0x26
```

3. Examples of non-U.S. English character mappings:

```
a_circumflex=0x83
a_umlaut=0x84
```

4. Examples of nonalphanumeric, noncontrol-key names mappings:

```
key_tab=0x100
key_btab=0x101
```

Implementation Specifics

This file is part of the AIX 3270 Host Connection Program/6000 (HCON).

Files

/usr/lib/hcon directory	Contains HCON files, including the **keynames** file.
/usr/lib/hcon/e789_ktbl file	Contains the default binary keyboard definition table.
/usr/lib/hcon/e789_ktbl.p file	Contains the source for the default binary keyboard definition table.
/usr/lib/hcon/func_names file	Contains HCON function names.

Related Information

Names defined in the **keynames** file are used as input to the **e789kdef** command and to customize the **e789_ktbl.p** file.

Customizing HCON in *Communication Concepts and Procedures* discusses the options available to you for customizing HCON keyboard definitions as well as key and function names.

HCON nls_names File Format

Purpose

Contains color and attribute name mappings used by HCON.

Description

The **/usr/lib/hcon/nls_names** file contains color and attribute name mappings used by the AIX 3270 Host Connection Program/6000 (HCON). The **nls_names** file maps color and attribute names to internal identifiers used by the **e789cdef** command. This command allows users to customize the 3270 field display attributes on their terminals for use by HCON emulation sessions.

The **nls_names** file allows color and attribute names to be translated into their equivalent in any language. Thus the character string that signifies a color or attribute name may change, but the internal identifier that the **e789cdef** command uses for the color or attribute remains constant.

The **nls_names** file contains mappings for:

- Terminal colors
- Terminal highlighting
- 3270 extended highlighting
- 3270 extended colors
- Status line
- Unformatted screen (with no field attributes)
- 3270 base field attributes.

Color and attribute name mappings are in the following format:

Name=InternalID [* *Comments*]

The mapping string *Name=InternalID* must not contain any spaces.

The *Name* variable can be translated into its equivalent form in any language. However, the length of the *Name* variable cannot be more than 36 characters.

The *InternalID* variable must not be changed.

Any line that starts with an * (asterisk) and any part of a line following an * are interpreted as comments by HCON.

Note: Changes to color and attribute names in the **nls_names** file affect the entire color customization process. All users must use the new names as input to the **e789cdef** command.

Examples

Following are examples of each type of entry in the **nls_names** file.

1. Examples of terminal color entries:

```
red=1           * The terminal color "red" has an id of 1
green=2         * The terminal color "green" has an id of 2
```

2. Example of a terminal highlighting entry:

```
blink=1          * The terminal hilite "blink" has an id of 1
```

3. Example of a 3270 extended highlighting entry:

```
underline=1      * The 3270 hilite "underline" has an id of 1
```

4. Examples of 3270 extended color entries:

```
defnorm=0        * The 3270 color "defnorm" has an id of 0
red=2            * The 3270 color "red" has an id of 2
```

5. Example of a 3270 base field attribute entry:

```
unorm=0 * The 3270 field attribute "unorm"
                 * has an id of 0
```

6. Example of a status line entry:

```
status=0         * The attribute "status" has an id of 0
```

7. Example of an unformatted screen entry:

```
unfrmt=1         * The attribute "unfrmt" has an id of 1
```

Following is an example of changing an entry to the equivalent name in another language:

8. To change the name of the terminal color red to the French name, `rouge`, replace `red` with `rouge` in the terminal color entries section of the **nls_names** file:

```
rouge=1          * The terminal color "rouge" has an id of 1
```

   To cause text to be displayed in red on the terminal screen, the user will now specify `rouge` as input to the **e789cdef** command.

Implementation Specifics

This file is part of the AIX 3270 Host Connection Program/6000 (HCON).

Files

/usr/lib/hcon directory	Contains HCON files, including the **nls_names** file.
/usr/lib/hcon/e789_ctbl file	Contains the default binary color definition table.
/usr/lib/hcon/e789_ctbl.p file	Contains the source for the default binary color definition table.

Related Information

Names defined in the **nls_names** file are used as input to the **e789cdef** command and to customize the **e789_ctbl.p** file.

Customizing HCON in *Communication Concepts and Procedures* discusses the options available to you for customizing HCON colors, keyboard definitions, and key and color names.

MH Alias File Format

Purpose

Defines aliases.

Description

An alias file contains lines that associate an alias name with an address or group of addresses. The Message Handler (MH) package reads both personal alias files and a systemwide alias file, the **/usr/lib/mh/MailAliases** file. Depending on the MH configuration, aliases may also be defined in the **/usr/lib/aliases** file (see the **sendmail** command).

The alias file name is an argument to several MH commands. These commands can be set automatically by entries in the **.mh_profile** file. Personal alias files can have any name, but must follow the format described here. The **/usr/lib/mh/MailAliases** file is the default alias file for systemwide aliases. This file is set up by a user with root user authority.

Specify your personal alias file in your **.mh_profile** file. If you do not do so, you must use the **–alias** flag each time you use an MH command that takes this flag.

Each line of an **.mh_alias** file has one of the following formats:

- *Alias* : *Address-Group*
- *Alias* ; *Address-Group*
- *<Alias-File*

The variables are described as follows:

Alias — The *Alias* string is a simple address.

Address — The *Address* variable must be a simple Internet-style address.

Group — The *Group* variable must be a group name (or number) from the **/etc/group** file.

Alias-File — The *File* variable must be an AIX file name. The MH package treats alias file names as case-sensitive. Alias expansion is case-sensitive as well.

The *Address-Group* variable can be either of the following:

AddressList — A list of addresses that make up a group.

<Alias-File — An AIX file to be read for more alias definitions.

The addresses in the *AddressList* variable are separated by commas.

Special Characters

\ (backslash) — You can continue an alias definition on the next line by ending the line to be continued with a \ (backslash) followed by a new-line character.

< (less than) — If a line starts with a < (less than) sign, MH reads the file specified after the less-than sign for more alias definitions. The reading is done recursively.

	If an address group starts with a < (less than) sign, MH reads the file specified after the less-than sign and adds the contents of that file to the address list for the alias.
= (equal)	If an address group starts with an = (equal) sign, MH consults the **/etc/group** file for the AIX group specified after an equal sign. The MH package adds each login name occurring as a member of the group to the address list for the alias.
+ (plus)	If an address group starts with a + (plus) sign, MH consults the **/etc/group** file to determine the group ID of the AIX group specified after a plus character. MH adds to the address list for the alias each login name occurring in the **/etc/passwd** file whose group ID is indicated by this group.
* (asterisk)	If an address group is defined by an * (asterisk), MH consults the **/etc/passwd** file and adds all login names with a user number greater than 200 (or the value set for everyone in the **/usr/lib/mh/mtstailor** file to the address list for the alias).

The following list explains how the system resolves aliases at posting time:

1. The system builds a list of all addresses from the message to be delivered, eliminating duplicate addresses.
2. If the draft originated on the local host, the system performs alias resolution for addresses that have no specified host.
3. For each line in the alias file, the system compares the alias with all existing addresses. If a match is found, the system removes the matched alias from the address list. It then adds each new address in the address group to the address list. The alias itself is not usually output. Instead, the address group to which the alias maps is output. If the alias is terminated with a ; (semicolon) instead of a : (colon), both the alias and the address are output in the correct form. (This correct form makes replies possible since MH aliases and personal aliases are unknown to the mail transport system.)

In pattern matching, a trailing * (asterisk) in an alias matches just about anything appropriate.

Example

The following example of an **.mh_alias** file illustrates some features:

```
</u/sarah/morealiases
systems:= systems
staff:+ staff
everyone:= *
manager:= harold@harold
project:= lance,mark@remote,peter,manager
```

The first line says that more aliases should be read from the `/u/sarah/morealiases` file. The name `systems` is defined as all users who are listed as members of the group `systems` in the **/etc/group** file, and `staff` is defined as all users whose group ID in the **/etc/passwd** file is equivalent to the `staff` group. Finally, `everyone` is defined as all users with a user ID in the **/etc/passwd** file greater than 200.

The name `manager` is defined as an alias for the user `harold@harold`. The name `project` is defined as an alias for the users `lance`, `mark@remote`, `peter`, and `manager`.

The Message Handler (MH) package reads alias files line by line. Therefore, references to an alias in a previous line will be implemented. References to an alias in a following line will not be implemented.

Implementation Specifics

This file format is part of Message Handler in BOS Extensions 1.

Files

/usr/lib/mh/MailAliases	Contains the defaults alias file for systemwide aliases, which is set up by a user with root user authority.
/usr/lib/aliases	Contains systemwide aliases for the sendmail command.

Related Information

The **ali** command, **conflict** command, **post** command, **sendmail** command, **whom** command.

The **aliases** file, **/etc/group** file, **/etc/passwd** file, **$HOME/.mh_profile** file.

PC Simulator Startup File Format

Purpose

Specifies options when starting PC Simulator.

Description

Whenever you start PC Simulator with the **pcsim** command, PC Simulator searches for a profile that specifies startup options. The profile used by PC Simulator is a pure ASCII text file that you can edit with any text editor.

You can specify a profile with the **profile** flag. Otherwise, PC Simulator searches for the default profile, **simprof**. A sample profile named **simprof**, included with PC Simulator, is located in the **/usr/lpp/pcsim/samples** directory.

You can have multiple profiles, for different users, or to start PC Simulator with different options in effect. PC Simulator searches for the specified profile first in the current working directory, then in the **$HOME** directory, and finally in the **/usr/lpp/pcsim** directory. If you have only one profile, you can copy the **simprof** sample profile to one of these directories, then edit it to set the options you want.

Even if PC Simulator finds a profile, it searches all three directories. It may, therefore, find more than one profile with the same file name. If this happens, option settings in the second profile override values specified for the same options in the first profile. Options set with flags from the command line override option settings found in a profile.

Example

A simulator profile resembles an AIXwindows defaults file. Options are listed by flag name, followed by a colon (:), then a parameter value. The **simprof** sample profile included with PC Simulator is similar to this example, except it includes no parameter values.

If an option is not listed, or no value is specified, PC Simulator starts with the default value for this option. A blank space between the colon and parameter value is optional. Any text following a pound sign (#) is a comment.

```
Cdrive        : /u/dos1/txt.fil  # select file /u/dos1/txt.fil
                                 # for fixed disk C:
Ddrive        : /u/dos2          # select directory /u/dos2
                                 # for fixed disk D:
permission    : 666              # read/write permissions to
                                 # all users for files saved
                                 # to fixed disk
Adiskette     : 3                # select 3.5-inch diskette drive
Bdiskette     :                  # no B diskette drive selected
dtime         : 5                # release diskette drive to
                                 # AIX after 5 seconds
autoRaise     :                  # window manager controls
                                 # window level
display       :                  # use default AIXwindows
                                 # server, unix:0
dmode         : V                # select VGA display mode
geometry      :                  # use default window size
                                 # & position, 720x494+152+265
iconBitmap    :                  # use default icon
iconGeometry  : =64X64+10+10     # size and position of icon
iconName      :                  # use default, pcsim
```

```
kbdmap      :                   # no file selected
name        : BUDGET            # name in window title bar
refresh     : 15                # refresh display every
                                # 15 milliseconds
font        :                   # use default font file, ROM10.500
warp        : true              # center initial mouse cursor
lpt1        : lp0               # emulate DOS lpt1 with AIX lp0
lpt2        :                   # none selected
lpt3        :                   # none selected
ptime       : 30                # print job file buffering
                                # time out after 30 seconds
xmemory     : 1024              # provide 1MB extended memory
```

Files

/usr/lpp/pcsim/samples/simprof Example startup profile.

Related Information

The **pcsim** command.

TCP/IP .3270keys File Format

Purpose

Defines a user keyboard mapping and colors for TELNET (3270).

Description

The **/etc/.3270keys** file allows a user to have a TELNET (3270) key mapping that differs from the default mapping. For example, you might want to change the ACTION key to the ENTER key.

If you are using a color display, you can also change this file to customize the colors for various 3270 display attributes. (The default mapping in the **3270.keys** file is generic. The **.3270keys** file allows users to tailor key mappings to the workstation keyboard and to select alternate colors for 3270 display attributes.) The **.3270keys** file exists as a hidden file in the user's home directory. The name of the hidden file begins with a . (dot).

Note: When remapping keys to customize your **.3270keys** file, keep in mind that you cannot map a 3270 function to the ESC key alone. You can specify the ESC key only in combination with another key.

Creating Your Own .3270keys File

The **/usr/lpp/tcpip/samples/3270keys.hft** file is a sample that can be used to create a **.3270keys** file. To create a **$HOME/.3270keys** file, copy the file to your home directory and make any modifications to it.

The following options can be used in the sequence field:

\b	backspace.
\s	space.
\t	tab.
\n	new line.
\r	return.
\e	escape.
^	mask next character with \037 (e.g. ^M).
~	set high order bit for next character.

The following are valid colors for 3270 display attributes:

- black
- blue
- red
- green
- white
- magenta
- cyan.

Note: You can also change the default key mappings by editing the **3270keys.hft** file for high function terminal mapping and the **3270.keys** file for ASCII terminal mapping.

Example

This example sets up two keys, the Backspace key and the Tab key.

```
      3270 Function  Sequence  Key
bind    backspace      "\b"      #backspace
bind    tab            "\t"      #tab
```

The # (pound sign) is used to indicate comments.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/etc/3270.keys Contains the default keyboard mapping for non-RT keyboards.

/etc/3270keys.hft Contains the default keyboard mapping for RT keyboards.

/usr/lpp/tcpip/samples/3270keys.hft
Contains a sample RT keyboard mapping. This file also contains directions for its use.

$HOME/.3270keys Defines the user's keyboard mapping for the TELNET Protocol.

/etc/tn3270.keys Contains the default keyboard mapping for the **tn3270** command.

Related Information

The **telnet, tn** command.

TCP/IP Domain Cache File Format

Purpose

Defines the root name server or servers for a domain name server host.

Description

The **cache** file is one of the DOMAIN data files and contains the addresses of the servers that are authoritative name servers for the root domain of the network. The name of this file is defined in the **named** boot file. If the host serves more than one domain, the cache file should contain an entry for the authoritative name server for each domain.

All entries in this file must be in Standard Resource Record Format. Valid resource records in this file are:

- Name Server (NS)
- Address (A)

Except for comments (starting with a ; (semicolon) and continuing to the end of the line), the resource records in the data files generally follow the format of the resource records that the **named** daemon returns in response to queries from resolver routines.

Example

The following examples show the various ways to use the cache data file. This example is valid for any name server or either of the two networks.

Network `abc` consists of:

- `gobi.abc`, the primary name server for the `abc` network, 192.9.201.2
- `mojave.abc`, a host machine, 192.9.201.6
- `sandy.abc`, secondary name server for the `abc` network and gateway between `abc` and `xyz`, 192.9.201.3.

Network `xyz` consists of:

- `kalahari.xyz`, primary name server for the `xyz` network, 160.9.201.4
- `lopnor.xyz`, a host machine, 160.9.201.5
- `sahara.xyz`, a host machine and cache-only name server for the `xyz` network, 160.9.201.13
- `sandy.xyz`, a secondary name server for the `xyz` network and gateway between `abc` and `xyz`, 160.9.201.3

Note: Note that `sandy`, a gateway host, is on both networks and also serves as secondary name server for both.

The following are sample entries in a Domain cache file on any of the name servers in either of the domains:

```
;
;cache file for all nameservers in both domains
;
; root name servers.
abc                     IN      NS      gobi.abc.
xyz                     IN      NS      kalahari.xyz.
gobi.abc.       3600000 IN      A       192.9.201.2
kalahari.xyz    3600000 IN      A       160.9.201.4
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/named.boot
: Contains a sample **named.boot** file. This file also contains directions for its use.

/etc//named.boot
: Defines how **named** initializes the DOMAIN name server file.

/usr/lpp/tcpip/samples/named.data
: Contains a sample **named.data** file. This file also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Data file, DOMAIN Reverse Data file, DOMAIN Local file.

Configuring Name Servers for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP Domain Data File Format

Purpose

Stores name resolution information for the **named** daemon.

Description

The hosts data file is one of the DOMAIN data files and contains name to address resolution mapping information for all machines in the name server's zone of authority. The name of the hosts data file is specified in the **named** boot file. This file should exist only on name servers that are designated as *primary* for a domain. There may be more than one hosts data file per primary name server.

All entries in this file must be in Standard Resource Record Format. Valid resource records in this file are:

- Start of Authority (SOA)
- Name Server (NS)
- Address (A)
- Mailbox (MB)
- Mail Exchanger (MX)
- Mail Group (MG)
- Mail Rename (MR)
- Canonical Name (CNAME)
- Well Known Services (WKS)
- Host Information (HINFO)

Except for comments (starting with a ; (semicolon) and continuing to the end of the line), the resource records in the data files generally follow the format of the resource records that the **named** daemon returns in response to queries from resolver routines.

Two **awk** scripts, **addrs.awk** and **hosts.awk**, are provided in the **/usr/lpp/tcpip/samples** directory to assist you in converting your existing **/etc/hosts** file to DOMAIN data files. These files also contain instructions for their use. Refer to these files for more information on the conversion.

Examples

The following examples show the various ways to use the Domain hosts data file. In these examples, two networks are represented: `abc` and `xyz`.

Network `abc` consists of:

- `gobi.abc`, the primary name server for the `abc` network, 192.9.201.2
- `mojave.abc`, a host machine, 192.9.201.6
- `sandy.abc`, secondary name server for the `abc` network and gateway between `abc` and `xyz`, 192.9.201.3.

Network `xyz` consists of:

- `kalahari.xyz`, primary name server for the `xyz` network, 160.9.201.4
- `lopnor.xyz`, a host machine, 160.9.201.5
- `sahara.xyz`, a host machine and cache-only name server for the `xyz` network, 160.9.201.13
- `sandy.xyz`, a secondary name server for the `xyz` network and gateway between `abc` and `xyz`, 160.9.201.3

Note: Note that `sandy`, a gateway host, is on both networks and also serves as secondary name server for both.

1. The primary host data file for network `abc`, stored on host `gobi.abc`, contains the following entries:

```
;
;primary host data file for abc — gobi.abc
;
@               IN      SOA     gobi.abc.  root.gobi.abc.  (
                                1:1     ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000;expire
                                86400   ;minimum
                                )
;name servers for abc
                IN      NS      gobi.abc.
;other name servers
                IN      NS      kalahari.xyz.
kalahari.xyz.   IN      A       160.9.201.4
;
;define local loopback host
localhost       IN      A       127.1
;
;define all hosts in abc
loopback        IN      CNAME   localhost.abc
gobi            IN      A       192.9.201.2
gobi-abc        IN      CNAME   gobi.abc
sandy           IN      A       192.9.201.3
                IN      WKS     192.9.201.3
udp tftp nameserver domain
                IN      WKS    192.9.201.3 tcp (
                               echo telnet smtp discard uucp-path
                               systat daytime netstat chargen ftp
                               time whois finger hostnames domain
```

```
                                        )
sandy-abc       IN      CNAME   sandy.abc
mojave          IN      A       192.9.201.6
                IN      HINFO   RISC-System/6000 AIX-3.1
mojave-abc      IN      CNAME   mojave.abc.
```

2. The primary host data file for network `xyz`, stored on host `kalahari.xyz`, contains the following entries:

```
;
;primary host data file for xyz - kalahari.xyz
;
@               IN      SOA     kalahari.xyz.  root.kalahari.xyz.
  (
                                1:1     ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000;expire
                                86400   ;minimum
                                )

;
;nameservers for xyz
;
                IN      NS      kalahari.xyz.
;
;other nameservers
                IN      NS      gobi.abc.
gobi.abc.       IN      A       192.9.201.2

;
;define local loopback host
localhost       IN      A       127.1

;
;define all hosts in xyz
loopback        IN      CNAME   localhost.xyz.
kalahari        IN      A       160.9.201.4
ns-xyz          IN      CNAME   kalahari.xyz.
kalahari-xyz    IN      CNAME   kalahari.xyz.
                IN      HINFO   RISC-System/6000 AIX-3.1
sahara          IN      A       160.9.201.13
                IN      WKS     160.9.201.13 (udp tftp nameserver
                                domain)
                IN      WKS     160.9.201.13 tcp (
                                echo telnet smtp discard uucp
                                path systat daytime netstat
                                chargen ftp time whois finger
                                hostnames domain
                                )
                IN      HINFO   RISC-System/6000 AIX-3.1
lopnor          IN      A       160.9.201.5
lopnor-xyz      IN      CNAME   lopnor.xyz.
                IN      HINFO   RISC-System/6000 AIX-3.1
sandy           IN      A       160.9.201.3
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/addrs.awk
Contains a sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.rev** file. This file also contains directions for its use.

/usr/lpp/tcpip/samples/hosts.awk
Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.data** file. This file also contains directions for its use.

/usr/lpp/tcpip/samples/named.boot
Contains a sample **named.boot** file. This file also contains directions for its use.

/etc/named.boot
Defines how **named** initializes the DOMAIN name server file.

/usr/lpp/tcpip/samples/named.data
Contains a sample **named.data** file. This file also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Reverse Data file, DOMAIN Cache file, DOMAIN Local file.

The TCP/IP Standard Resource Record Format.

Configuring Name Servers for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP Domain Local Data File Format

Purpose

Defines the local loopback information for **named** on the name server host.

Description

The local data file is one of the DOMAIN data files and contains local loopback information for the name server host. The name of the DOMAIN local data files is specified in the **named** boot file.

All entries in this file must be in Standard Resource Record Format. Valid resource records in local data file are:

- Start of Authority (SOA)
- Name Server (NS)
- Pointer (PTR)

The records in the DOMAIN data files are called resource records. Except for comments (starting with a ; (semicolon) and continuing to the end of the line), the resource records in the data files generally follow the format of the resource records that the **named** daemon returns in response to queries from resolver routines.

Examples

The following examples show the various ways to use the Domain local data file. In these examples, two networks are represented: `abc` and `xyz`.

Network `abc` consists of:

- `gobi.abc`, the primary name server for the `abc` network, 192.9.201.2
- `mojave.abc`, a host machine, 192.9.201.6
- `sandy.abc`, secondary name server for the `abc` network and gateway between `abc` and `xyz`, 192.9.201.3.

Network `xyz` consists of:

- `kalahari.xyz`, primary name server for the `xyz` network, 160.9.201.4
- `lopnor.xyz`, a host machine, 160.9.201.5
- `sahara.xyz`, a host machine and cache–only name server for the `xyz` network, 160.9.201.13
- `sandy.xyz`, a secondary name server for the `xyz` network and gateway between `abc` and `xyz`, 160.9.201.3

Note: Note that `sandy`, a gateway host, is on both networks and also serves as secondary name server for both.

1. The **named.abclocal** file stored on `gobi.abc` contains the following entries:

```
;
;primary reverse file for local 127 network
;
@               IN      SOA     gobi.abc.  root.gobi.abc.
                                (
                                1:1     ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000;expire
                                86400   ;minimum
                                )
                IN      NS      gobi.abc.
1               IN      PTR     localhost.
```

2. The **named.xyzlocal** file stored on `kalahari.xyz` contains the following entries:

```
;
;primary reverse file for local 127 network
;
@               IN      SOA     kalahari.xyz.  root.kalahari.xyz.

                                (
                                1:1     ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000;expire
                                86400   ;minimum
                                )
                IN      NS      kalahari.xyz.
1               IN      PTR     localhost.
```

3. The **named.seclocal** file stored on `sandy` contains the following entries:

```
;
;primary reverse file for local 127 network
;
@               IN      SOA     sandy.abc.  root.sandy.abc.
                                (
                                1:1     ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000;expire
                                86400   ;minimum
                                )
                IN      NS      sandy.abc.
1               IN      PTR     localhost.
```

4. The **named.calocal** file stored on `sahara.xyz` contains the following entries:

```
;
;primary reverse file for local 127 network
;
@               IN      SOA     sahara.xyz.  root.sahara.xyz.
                                (
                                1:1     ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000;expire
                                86400   ;minimum
                                )
                IN      NS      sahara.xyz.
1               IN      PTR     localhost.
```

Implementation Specifics

This file is part of TCP/IP Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/named.boot
Contains a sample **named.boot** file. This file also contains directions for its use.

/etc/named.boot
Defines how **named** initializes the DOMAIN name server file.

/usr/lpp/tcpip/samples/named.data
Contains a sample **named.data** file. This file also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Data file, DOMAIN Reverse Data file, DOMAIN Cache file.

Configuring Name Servers for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures.*

TCP/IP Domain Reverse Data File Format

Purpose

Stores reverse name resolution information for the **named** daemon.

Description

The Reverse Data File is one of the DOMAIN data files and contains address to name resolution mapping information for all machines in the name server's zone of authority. The name of the reverse hosts data file is specified in the **named** boot file. There may be more than one reverse hosts data file per primary name server.

All entries in this file must be in Standard Resource Record Format. Valid resource records in this file are:

- Start of Authority (SOA)
- Name Server (NS)
- Pointer (PTR)

Except for comments (starting with a ; (semicolon) and continuing to the end of the line), the resource records in the data files generally follow the format of the resource records that the **named** daemon returns in response to queries from resolver routines.

Two **awk** scripts, **addrs.awk** and **hosts.awk**, are provided in the **/usr/lpp/tcpip/samples** directory to assist you in converting your existing **/etc/hosts** file to **named** data files. These files also contain instructions for their use. Refer to these files for more information on the conversion.

Examples

The following examples show the various ways to use the Domain reverse data file. In these examples, two networks are represented: `abc` and `xyz`.

Network `abc` consists of:

- `gobi.abc`, the primary name server for the `abc` network, 192.9.201.2
- `mojave.abc`, a host machine, 192.9.201.6
- `sandy.abc`, secondary name server for the `abc` network and gateway between `abc` and `xyz`, 192.9.201.3.

Network `xyz` consists of:

- `kalahari.xyz`, primary name server for the `xyz` network, 160.9.201.4;
- `lopnor.xyz`, a host machine and cache–only name server for the `xyz` network, 160.9.201.5
- `sahara.xyz`, a host machine, 160.9.201.13
- `sandy.xyz`, a secondary name server for the `xyz` network and gateway between `abc` and `xyz`, 160.9.201.3

Note: Note that `sandy`, a gateway host, is on both networks and also serves as secondary name server for both.

1. The reverse data file for `gobi.abc`, primary name server for network `abc`, contains these entries:

```
;
;primary reverse host data file for abc – gobi.abc
;
@                       IN      SOA     gobi.abc.  root.gobi.abc.
  (
                                        1:1     ;serial
                                        3600    ;refresh
                                        600     ;retry
                                        3600000;expire
                                        86400   ;minimum
                                        )
;nameservers for abc
                                IN      NS      gobi.abc.
;other nameservers
                                IN      NS      kalahari.xyz.
4.201.9.160.in–addr.arpa        IN      PTR     kalahari.xyz
;
;define all hosts in abc
2                               IN      PTR     gobi.abc.
3                               IN      PTR     sandy.abc.
6                               IN      PTR     mojave.abc.
```

2. The reverse data file for `kalahari.xyz`, primary name server for network `xyz`, contains these entries:

```
;
;primary reverse host data file for xyz – kalahari.xyz
;
@               IN      SOA     kalahari.xyz. root.kalahari.xyz.
 (
                                1:1     ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000;expire
                                86400   ;minimum
                                )
;nameservers for xyz
                                IN      NS      kalahari.xyz.
;other nameservers
                                IN      NS      gobi.abc.
2.201.9.192.in–addr.arpa        IN      PTR     gobi.abc
;
;define all hosts in xyz
2.201                           IN      PTR     kalahari.xyz.
13.201                          IN      PTR     sahara.xyz.
5.201                           IN      PTR     lopnor.xyz.
3.201                           IN      PTR     sandy.xyz.
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/addrs.awk
Contains a sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.rev** file. This file also contains directions for its use.

/usr/lpp/tcpip/samples/hosts.awk
Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.data** file. This file also contains directions for its use.

/usr/lpp/tcpip/samples/named.boot
Contains a sample **named.boot** file. This file also contains directions for its use.

/etc/named.boot
Defines how **named** initializes the DOMAIN name server file.

/usr/lpp/tcpip/samples/named.data
Contains a sample **named.data** file. This file also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Data file, DOMAIN Cache file, DOMAIN Local file.

The Standard Resource Record Format.

Configuring Name Servers for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP ftpusers File Format

Purpose

Specifies local user names that cannot be used by remote FTP clients.

Description

This file contains a list of local user names that the **ftpd** server does *not* allow to be used by remote File Transfer Protocol (FTP) clients. The format of the **/etc/ftpusers** file is a simple list of user names that also appear in the **/etc/passwd** file.

Entries to this file can be made using the System Management Interface Tool (SMIT) or using the **ruser** command.

Example

The following are sample entries in an **ftpusers** file:

```
root
guest
ftp
joan
UUCP
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/ftpusers	Contains a list of user names that are not to be used by the remote File Transfer Protocol users.

Related Information

The **ftpd** daemon.

The **ruser** command.

Understanding the File Transfer Protocol (FTP) and Understanding the SMIT Interface to TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP gated.conf File Format

Purpose

Contains configuration information for the **gated** daemon.

Description

The **/etc/gated.conf** file contains configuration information that is read by the **gated** daemon at initialization time. This file contains stanzas that control tracing options, select routing protocols, manage routing information, and manage independent system routing.

Stanzas can appear in any order in the **gated.conf** file. The following sections illustrate the format of each stanza.

Controlling Trace Output

The option that controls trace output is read during the initialization of the **gated** daemon and whenever the **gated** daemon receives a **SIGHUP** signal. This option is overridden at initialization time if trace flags are specified to the **gated** daemon on the command line.

Specifying the Level of Trace Output

The **traceflags** stanza is in the following format and tells the **gated** daemon what level of trace output you want:

`traceflags` *`Flag [Flag Flag . . . ]`*

The valid flags for tracing are as follows:

internal Logs all internal errors and interior routing errors.

external Logs all external errors due to EGP, exterior routing errors, and EGP state changes.

route Logs all routing changes.

egp Traces all EGP packets sent and received.

update Logs all routing updates sent.

rip Traces all RIP packets received.

hello Traces all HELLO packets received.

icmp Traces all ICMP direct packets received.

snmp Traces all SNMP transactions.

stamp Prints a time stamp to the log file every 10 minutes.

general Combines the **internal**, **external**, **route**, and **egp** flags.

all Enables all of the listed trace flags.

If more than one **traceflags** stanza is used, the trace flags specified in all stanzas are enabled.

Selecting Routing Protocols

This section explains the configuration options for routing protocols. These options provide the **gated** daemon with instructions on how to manage routing for each protocol.

Note: All references to point–to–point interfaces in the **gated** configuration file must use the address specified by the *Destination* parameter.

Using the gated Daemon with the RIP Protocol

The following stanza tells the **gated** daemon how to perform the Routing Information Protocol (RIP) routing protocol:

RIP { yes | no } [supplier | pointopoint] [quiet] [gateway *HopCount*]

A list of the arguments to the RIP stanza follows:

yes Performs the RIP protocol, processing all incoming RIP packets and supplying RIP information every 30 seconds only if there are two or more network interfaces.

no Specifies that the RIP protocol not be performed.

supplier Performs the RIP protocol, processing all incoming RIP packets and forcing the supply of RIP information every 30 seconds no matter how many network interfaces are present.

pointopoint Performs the RIP protocol, processing all incoming RIP packets and forcing the supply of RIP information every 30 seconds no matter how many network interfaces are present. When this argument is specified, RIP information is not sent out in a broadcast packet. The RIP information is sent directly to the gateways listed in the sourceripgateways stanza.

quiet Processes all incoming RIP packets, but does not supply any RIP information no matter how many network interfaces are present.

gateway *HopCount*
Processes all incoming RIP packets, supplying RIP information every 30 seconds and announcing the default route (0.0.0.0) with a metric specified by the *HopCount* variable. The metric should be specified in a value that represents a RIP hop count.

With this option set, all other default routes coming from other RIP gateways are ignored. The default route is only announced when actively peering with at least one EGP neighbor and therefore should only be used when EGP is used.

If no **RIP** stanza is specified, RIP routing is not performed.

Using the gated Daemon with the HELLO Protocol

The following stanza configures the Defense Communications Network Local–Network Protocol (HELLO) routing protocol for the **gated** daemon:

HELLO { yes | no } [supplier | pointopoint] [quiet] [gateway *Milliseconds*]

The *Argument* variable parallels the RIP arguments, with some minor differences.

A list of the arguments to the HELLO stanza follows:

yes Performs the HELLO protocol, processing all incoming HELLO packets and supplying HELLO information every 15 seconds only if there are two or more network interfaces.

no Specifies that this gateway does not perform the HELLO protocol.

supplier Performs the HELLO protocol, processing all incoming HELLO packets and forcing a supply of HELLO information every 15 seconds no matter how many network interfaces are present.

pointopoint Performs the HELLO protocol, processing all incoming HELLO packets and forcing a supply of HELLO information every 15 seconds no matter how many network interfaces are present.

When this argument is specified, HELLO information is not sent out in a broadcast packet. The HELLO information is sent directly to the gateways listed in the sourcehellogateways stanza.

quiet Processes all incoming HELLO packets, but does not supply any HELLO information regardless of the number of network interfaces present.

gateway *Milliseconds*

Processes all incoming HELLO packets, supplying HELLO information every 15 seconds and announcing the default route (0.0.0.0) with a time delay specified by the *Millisecond* variable. The time delay should be a numeric value specified in milliseconds.

The default route is only announced when actively peering with at least one EGP neighbor. Therefore, this stanza should only be used when running EGP.

If no HELLO stanza is specified, HELLO routing is not performed.

Using the gated Daemon with the EGP Protocol

The following stanzas specify the information necessary for the **gated** daemon to use the Exterior Gateway Protocol (EGP).

EGP { yes | no } This stanza allows the processing of EGP by the **gated** daemon to be turned on or off. The arguments are interpreted as follows:

yes Performs all EGP operations.

no Specifies that no EGP processing should be performed.

Note: EGP processing takes place by default. If no EGP stanza is specified, all EGP operations take place.

autonomous system *Number*

When the **gated** daemon performs the EGP protocol, this stanza must be used to specify the independent (autonomous) system number. If this number is not specified, the **gated** daemon exits immediately with an error message.

egpmaxacquire *Number*

When the **gated** daemon uses the EGP protocol, this stanza specifies the number of EGP peers with whom the **gated** daemon uses EGP. The *Number* variable must be a value greater than 0 and less than or equal to the number of EGP neighbors specified, or the **gated** daemon exits immediately. If this stanza is omitted, all EGP neighbors are acquired.

egpneighbor *Gateway*

When the gated daemon uses the EGP protocol, this stanza specifies an EGP neighbor. The Gateway variable is the address of an EGP neighbor, and can be expressed in symbolic name (joe.austin.ibm.com) or dotted decimal (192.5.8.1) format. Each EGP neighbor will be acquired in the order listed in this file.

metricin *Metric*

egpmetricout *EGPMetric*

nogendefault

acceptdefault

defaultout *EGPMetric*

validate

intf *Interface*

sourcenet *Network*

gateway *Gateway*

When the **gated** daemon performs the EGP protocol, this stanza specifies with whom the **gated** daemon is to perform EGP. The gateway specified by the *Gateway* variable can be either a host address in Internet dotted decimal notation or a symbolic name from the **/etc/hosts** file.

Each EGP neighbor should have its own egpneighbor stanza and is acquired in the order listed in the **gated.conf** file.

The arguments to the egpmaxaquire *Number* stanza have the following definitions:

metricin *Metric* Specifies the internal time delay to be used as a metric for all of the routes learned from this neighbor. The *Delay* variable should be specified as a time delay from 0 to 30000. If this keyword and the **validate** keyword are not used, the internal metric used is the EGP distance multiplied by 100.

egpmetricout *EGPMetric*
Sets the EGP distance used for all nets advertised to this neighbor. The *EGPMetric* variable should be specified as an EGP distance in the range of 0 to 255. If this keyword is not specified, the internal time delay for each route is converted to an EGP distance by division by 100, with distances greater than 255 being set to 255.

nogendefault Specifies that this neighbor should not be considered for the internal generation of a default when the **RIP gateway** or the **HELLO gateway** argument is used. If not specified, the internal default is generated when actively peering with this neighbor.

acceptdefault Indicates that the default route (net 0.0.0.0) should be considered valid when received from this neighbor. If this keyword is not specified, on reception of the default route the **gated** daemon displays a warning message and ignores the route.

defaultout *EGPMetric*

Specifies that the internally generated default may be passed to this EGP neighbor at the specified distance. The distance should be specified as an EGP distance from 0 to 255. A default route learned from another gateway is not propagated to an EGP neighbor.

Without this keyword, no default route is passed through EGP. The **acceptdefault** keyword should not be specified when the **defaultout** keyword is used. The EGP metric specified in the **egpmetricout** keyword does not apply when the **defaultout** keyword is used. The default route always uses the metric specified by the **defaultout** keyword.

validate Specifies that all networks received from this EGP neighbor must be defined in a validAS stanza that also specifies the autonomous system of this neighbor. Networks without a validAS stanza are ignored after a warning message is printed.

intf *Interface* Defines the interface used to send EGP packets to this neighbor. This keyword is only used when there is no common net or subnet with this EGP neighbor. This keyword is present for testing purposes and does not imply correct operation when peering with an EGP neighbor that does not share a common net or subnet.

sourcenet *Network*

Specifies the source network to be used in EGP poll packets sent to this neighbor. If this keyword is not specified, the network (not subnet) of the interface used to communicate with this neighbor is used. This keyword is present for testing purposes and does not imply correct operation when used.

gateway *Gateway*

Specifies the gateway to use when installing routes learned from an EGP neighbor on a different network. Normally these routes would be ignored.

Managing Routing Information

The following configuration file stanzas determine how the **gated** daemon handles both incoming and outgoing routing information.

Specifying RIP or HELLO Gateways to Which the gated Daemon Listens

When the following stanzas are specified, the **gated** daemon only listens to RIP or HELLO information, respectively, from these RIP or HELLO gateways:

trustedripgateways *Gateway* [*Gateway Gateway* . . .]
trustedhellogateways *Gateway* [*Gateway Gateway* . . .]

The *Gateway* variable can be either an Internet address in dotted decimal notation, which avoids confusion, or a symbolic name from the **/etc/hosts** file. Note that the propagation of routing information is not restricted by this stanza.

Specifying Gateways for the gated Daemon to Send RIP or HELLO Information

With the following stanzas, the **gated** daemon sends RIP or HELLO information directly to the gateways specified:

sourceripgateways *Gateway* [*Gateway Gateway* . . .]
sourcehellogateways *Gateway* [*Gateway Gateway* . . .]

If the **pointopoint** argument is specified in the RIP or HELLO stanzas defined earlier, the **gated** daemon sends only RIP or HELLO information to the specified gateways and does *not* send out any information using the broadcast address.

If the **pointopoint** argument is not specified in those stanzas and the **gated** daemon is supplying RIP or HELLO information, the **gated** daemon sends information to the specified gateways and also broadcasts information using a broadcast address.

Turning Routing Protocols On and Off by Interface

The following stanzas turn routing protocols on and off by interface:

noripoutinterface *InterfaceAddress* [*InterfaceAddress InterfaceAddress* . . .]
nohellooutinterface *InterfaceAddress* [*InterfaceAddress InterfaceAddress* . . .]
noripfrominterface *InterfaceAddress* [*InterfaceAddress InterfaceAddress* . . .]
nohellofrominterface *InterfaceAddress* [*InterfaceAddress InterfaceAddress* . . .]

A **noripfrominterface** or **nohellofrominterface** stanza means that no RIP or HELLO information is accepted coming into the listed interfaces from another gateway.

A **noripoutinterface** or **nohellooutinterface** stanza means that no RIP or HELLO knowledge is sent out of the listed interfaces. The *InterfaceAddress* variable should be an Internet address in dotted decimal notation.

Stopping the gated Daemon from Timing Out Interfaces

The following stanza stops the **gated** daemon from timing out the interfaces whose addresses are listed in Internet dotted decimal notation by the *InterfaceAddress* arguments. These interfaces are always considered up and working.

passiveinterfaces *InterfaceAddress* [*InterfaceAddress InterfaceAddress* . . .]

This stanza is used because the **gated** daemon times out an interface when no RIP, HELLO, or EGP packets are being received on that particular interface, in order to dynamically determine if an interface is functioning properly.

PSN interfaces send a RIP or HELLO packet to themselves to determine if the interface is properly functioning, since the delay between EGP packets may be longer than the interface

time out. Interfaces that have timed out automatically have their routes re–installed when routing information is again received over the interface.

If the **gated** daemon is not a RIP or HELLO supplier, no interfaces are aged and the **passiveinterfaces** stanza automatically applies to all interfaces.

Specifying an Interface Metric

The following stanza allows the specification of an interface metric for the listed interface:

interfacemetric *InterfaceAddress Metric*

On systems that support interface metrics, this stanza overrides the kernel's metric. On systems that do not support an interface metric, this feature allows one to be specified.

The interface metric is added to the true metric of each route that comes in with routing information from the listed interface. The interface metric is also added to the true metric of any information sent out through the listed interface. The metric of directly attached interfaces is also set to the interface metric, and routing information broadcast about directly attached nets is based on the interface metric specified.

The **interfacemetric** stanza is required for each interface on which an interface metric is desired.

Providing Hooks for Fallback Routing

The following stanza provides hooks for fallback routing in the **gated** daemon.

fixedmetric *InterfaceAddress Protocol* **rip** | **hello** *Metric*

If this stanza is used, all routing information sent out by the specified interface has a metric specified by the *Metric* variable. For RIP, specify the metric as a RIP hop count from 0 to infinity. For HELLO, specify the metric as a HELLO delay in milliseconds from 0 to infinity. Any route that has a metric of infinity is left as infinity.

Note: Fixed metrics should be used with extreme caution.

Specifying Information to Be Ignored

The following stanza indicates that any information regarding the *Network* variable that comes in by means of the specified protocols and from the specified interfaces is ignored:

donotlisten *Network* **intf** *Address* [*Address . . .*] **proto rip** | **hello**
donotlistenhost *Host* **intf** *Address* [*Address . . .*] **proto rip** | **hello**

The **donotlisten** stanza contains the following information: the **donotlisten** keyword, followed by a network number specified by the *Network* variable, which should be in dotted decimal notation, followed by the **intf** keyword. Next is a list of interfaces in dotted decimal notation, then the **proto** keyword, followed by the **rip** or **hello** keyword.

The **all** keyword can be used after the **intf** keyword to specify all interfaces on the system. For example:

```
donotlisten 10.0.0.0 intf 128.84.253.200 proto rip
```

means that any RIP information about network `10.0.0.0` coming in by interface `128.84.253.200` will be ignored. One stanza is required for each network on which this restriction is desired. In addition:

```
donotlisten 26.0.0.0 intf all proto rip hello
```

means that any RIP and HELLO information about net `26.0.0.0` coming in through any interface is ignored.

The **donotlistenhost** stanza is defined in the same way, except that a host address is provided instead of a network address. Restrictions on routing updates are applied to the specified host route indicated by the specified routing or protocols.

Specifying Network or Host Information to Which the gated Daemon Listens

The following stanzas indicate that **gated** daemon that should listen to specified protocols and gateways:

listen *Network* **gateway** *Address* [*Address . . .*] **proto rip** | **hello**
listenhost *Host* **gateway** *Address* [*Address . . .*] **proto rip** | **hello**

The **listen** and **listenhost** stanzas specify to listen only to information about a network or host on the specified protocol or protocols and from the listed gateways.

These stanzas read as follows: the **listen** or **listenhost** keyword is followed by a network or host address, respectively, in dotted decimal notation. Next is the **gateway** keyword with a list of gateways in dotted decimal notation, and then the **proto** keyword followed by the **rip** or **hello** keyword. For example:

```
listen 128.84.0.0 gateway 128.84.253.3 proto hello
```

indicates that any HELLO information about network `128.84` that comes in through gateway `128.84.253.3` is accepted. Any other information about network `128.84` from any other gateway is rejected. One stanza is needed for each net to be restricted.

Also, the stanza:

```
listenhost 26.0.0.15 gateway 128.84.253.3 proto rip
```

means that any information about host `26.0.0.15` must come through RIP from gateway `128.84.253.3`. All other information regarding this host is ignored.

Restricting Announcements of Networks and Hosts

The following stanzas allow restriction of the networks and hosts that are announced and the protocols that announce them:

announce *Network InterfaceAddress* [*Address . . .*] *Protocol Type* [*EGPMetric*]
announcehost *Host InterfaceAddress Protocol Type* [*EGPMetric*]
noannounce *Network InterfaceAddress* [*Address . . .*] *Protocol Type* [*EGPMetric*]
noannouncehost *Host InterfaceAddress Protocol Type* [*EGPMetric*]

The **announce{host}** and **noannounce{host}** stanzas cannot be used together on the same interface. With the **announce{host}** stanza, the **gated** daemon only announces the nets or hosts that have an associated **announce** or **announcehost** stanza with the appropriate protocol.

With the **noannounce{host}** stanza, the **gated** daemon announces everything *except* those nets or hosts that have an associated **noannounce** or **noannouncehost** stanza. These stanzas provide a choice of announcing only what is on the announce list or everything except those nets on the **noannounce** list on an individual basis.

The arguments are the same as in the **donotlisten** stanza except **egp** may be specified by the *Proto* variable. The value of the *Type* variable can be **rip**, **hello**, **egp**, or any combination of the three. When **egp** is specified in the *Proto* field, an EGP metric must be specified. This is the metric at which the **gated** daemon announces the listed net through EGP.

Note that these are not static route entries. These restrictions only apply if the net or host is learned through one of the routing protocols. If a restricted network suddenly becomes unreachable and goes away, announcement of this net stops until it is learned again.

Only one **announce{host}** or **noannounce{host}** stanza may be specified for each network or host. A network or host cannot, for instance, be announced through HELLO for one interface and through RIP for another.

Some example **announce** stanzas might include:

```
announce 128.84 intf all proto rip hello egp egpmetric 0
announce 10.0.0.0 intf all proto rip
announce 0.0.0.0 intf 128.84.253.200 proto rip
announce 35.0.0.0 intf all proto rip egp egpmetric 3
```

With only these four **announce** stanzas in the configuration file, the **gated** process only announces these four nets. Network `128.84.0.0` is announced through RIP and HELLO to all interfaces and through EGP with a metric of `0`. Network `10.0.0.0` is announced through RIP to all interfaces.

Network `0.0.0.0` (default) is announced by RIP out interface `128.84.253.200` only. Network `35.0.0.0` is announced through RIP to all interfaces and announced through EGP with a metric of `3`. These are the only nets that are broadcast by this gateway.

Once the first **announce** stanza is specified, only the networks with **announce** stanzas are broadcast, including local subnets. Once an **announce{host}** or **noannounce{host}** stanza has an **all** keyword specified after an **intf** keyword, that stanza is applied globally and the option of having individual interface restrictions is lost.

If no routing announcement restrictions are desired, **announce** stanzas should not be used. All information learned is then propagated out. That announcement has no affect on the information to which the **gated** daemon listens.

Any network that does not have an **announce** stanza is still added to the kernel routing tables, but it is not announced through any of the routing protocols. To stop networks from being added to the kernel, the **donotlisten** stanza may be used.

As another example:

```
announce 128.84 intf 128.59.2.1 proto rip
noannounce 128.84 intf 128.59.1.1 proto rip
```

indicates that on interface `128.59.2.1`, only information about network `128.84.0.0` is announced through RIP, but on interface `128.59.1.1`, all information is announced, except `128.84.0.0` through RIP.

The stanzas:

```
noannounce 128.84 intf all proto rip hello egp egpmetric 0
noannounce 10.0.0.0 intf all proto hello
```

mean that except for the two specified nets, all networks are propagated. Specifically, network `128.84.0.0` is not announced on any interface through any protocols. Knowledge of network `128.84.0.0` is not sent anywhere. Network `10.0.0.0` is not announced through HELLO to any interface.

The second stanza also implies that network `10.0.0.0` is announced to every interface through RIP. This net is also broadcast through EGP with the metric specified in the **defaultegpmetric** stanza.

Defining a Default EGP Metric

The following stanza defines a default Exterior Gateway Protocol (EGP) metric to use when there are no routing restrictions:

defaultegpmetric *Number*

Without routing restrictions, the **gated** daemon announces all networks learned through HELLO or RIP through EGP with this specified default EGP metric. If this stanza is not used, the default EGP metric is set to 255, which causes any EGP advertised route of this nature to be ignored.

When there are no routing restrictions, any network with a direct interface is announced through EGP with a metric of 0. Note that this does not include subnets, but only the non–subnetted network.

Defining a Default Gateway

The following stanza defines a default gateway, which is installed in the kernel routing tables during initialization and re–installed whenever information about the default route is lost:

defaultgateway *Gateway* [*Metric*] *Protocol* [**active** | **passive**]

This route is installed with a time delay equivalent to a RIP metric of 15, unless another metric is specified with the *Metric* variable.

If the **RIP gateway** or **HELLO gateway** argument is in use, this default route is deleted.

An active default route is overridden by any other default route learned through another routing protocol. A **passive** default route is only overridden by a default route with a lower metric. In addition, an **active** default route is not propagated in routing updates, while a passive default route is propagated.

The gateway specified by the *Gateway* variable should be an address in Internet dotted decimal notation. The *Metric* variable is optional and should be a time delay from 0 to 30000. If a *Metric* is not specified, a time delay equivalent to a RIP metric of 15 is used.

The *Protocol* variable should be either **rip**, **egp**, or **hello**. The *Protocol* variable initializes the protocol by which the route was learned. In this case the *Protocol* variable is unused but remains for consistency.

Installing a Static Route

The following stanzas install static routes:

net *NetworkAddress* **gateway** *Address* **metric** *HopCount* **rip** | **egp** | **hello**
host *HostAddress* **gateway** *Address* **metric** *HopCount* **rip** | **egp** | **hello**

The **net** and **host** stanzas install a static route to the network specified by the *NetworkAddress* variable or the host specified by the *HostAddress* variable through a gateway specified by the *Address* variable at a metric specified by the *HopCount* variable learned through RIP, HELLO, or EGP. Again, dotted decimal notation should be used for the addresses. These routes are installed in the kernel's routing table and are never affected by any other gateway's RIP or HELLO announcements. The protocol by which they were learned is important if the route is to be announced through EGP.

If the protocol is RIP or HELLO and there are no routing restrictions, then this route is announced by EGP with a metric of **defaultegpmetric**. If the protocol keyword is **egp** and there are no routing restrictions, then this route is announced by EGP with a metric specified by the *HopCount* variable.

Restricting EGP Announcements

The following stanza provides a *soft restriction* to the **gated** daemon:

egpnetsreachable *Network* [*Network Network* . . .]

It cannot be used when the **announce** or **noannounce** stanzas are used. With no restrictions, the **gated** daemon announces all routes learned from RIP and HELLO through EGP. The **egpnetsreachable** stanza restricts EGP announcement to those networks listed in the stanza.

The metric used for routes learned through HELLO and RIP is the value given in the **defaultegpmetric** stanza. If this stanza does not specify a value, the value is set to 255. With the **egpnetsreachable** stanza, unique EGP metrics cannot be set for each network. The **defaultegpmetric** stanza is used for all networks except those that are directly connected, which use a metric of 0.

Specifying Invalid Networks

The following stanza appends to the **gated** daemon's list of martian networks, which are those that are known to be invalid and should be ignored:

martiannets *Network* [*Network Network . . .*]

When the **gated** daemon receives information about one of these networks through any means, it immediately ignores it. If external tracing is enabled, a message is printed to the trace log. Multiple occurrences of the **martiannets** stanza accumulate.

The initial list of martian networks provided by the **gated** daemon contains the following networks: 127.0.0.0, 128.0.0.0, 191.253.0.0, 192.0.0.0, 223.255.255.0, and 224.0.0.0.

Managing Autonomous System Routing

In the internal routing tables, the **gated** daemon maintains the autonomous system number from which each route was learned. Independent (autonomous) systems are used only when an exterior routing protocol is in use, in this case EGP.

Routes are tagged with the autonomous system number of the EGP peer from which they were learned. Routes learned through the interior routing protocols, RIP and HELLO, are tagged with the autonomous system number specified in the **autonomoussystem** stanza of the **gated.conf** file.

Note: The **gated** server does not normally propagate routes learned from exterior routing protocols to interior routing protocols, since some gateways do not have adequate validation of routing information they receive. Some of the following stanzas allow exterior routes to be propagated through interior protocols. Therefore, it is imperative that utmost care be taken when allowing the propagation of exterior routes.

The following stanzas provide limited control over routing based on autonomous system numbers.

Validating Networks from an Independent (Autonomous) System

The following stanza is used for validation of networks from a certain independent system:

validAS *Network* **AS** *System* **metric** *Number*

When an EGP update is received from a neighbor that has the **validate** keyword specified in the associated **egpneighbor** stanza, a **validAS** stanza is searched for that defines the network and the autonomous system number of the EGP neighbor.

If the appropriate **validAS** stanza is located, the network is considered for addition to the routing table with the specified metric. If a **validAS** stanza is not located, a warning message is printed and the network is ignored.

A network may be specified in several **validAS** stanzas as being associated with several different autonomous systems.

Controlling Exchange of Routing Information Between Autonomous Systems

The following stanzas control routing information exchange:

announcetoAS *AutonomousSystem1* **ASlist** *AutonomousSystem2* [*AutonomousSystem3* . . .]
noannouncetoAS *AutonomousSystem1* **ASlist** *AutonomousSystem2* [*AutonomousSystem3* . . .]

The **announcetoAS** and **noannouncetoAS** stanzas control the exchange of routing information between different autonomous (independent) systems. Normally the **gated** daemon does not propagate routing information between independent systems.

The exception to this is that routes learned from the **gated** daemon's own independent system through RIP and HELLO are propagated through EGP. These stanzas allow information learned through EGP from one autonomous system to be propagated through EGP to another autonomous system or through RIP and HELLO to the **gated** daemon's own autonomous system.

If the **announcetoAS** stanza is specified, information learned through EGP from autonomous systems *AS1*, *AS2*, *AS3*, and so on, is propagated to autonomous system *AS0*. If the **gated** daemon's own autonomous system, as specified in the **autonomoussystem** stanza, is specified as *AS0*, this information is propagated through RIP and HELLO. Routing information from autonomous systems not specified in the **ASlist** are not propagated to autonomous system *AS0*.

If the **noannouncetoAS** stanza is specified, information learned through EGP from all autonomous systems except *AS1*, *AS2*, *AS3*, and so on is propagated to autonomous system *AS0*. If the **gated** daemon's own autonomous system is specified as *AS0*, this information is not propagated through RIP and HELLO.

Only one **announcetoAS** or **noannounceAS** stanza may be specified for each target autonomous system.

Example

Setting Up a gated.conf File for EGP Routing

An example **gated.conf** file for a **gated** server that performs only EGP routing might contain the following entries:

The following three lines specify which protocol will be running. RIP and HELLO do not run. EGP does run.

```
RIP     no
HELLO   no
EGP     yes
#
```

The **traceflags** stanza tells what level of trace output is desired:

internal	Logs all internal error and interior routing errors
external	Logs all external errors due to EGP, exterior routing errors and EGP state changes
route	Logs all routing changes
egp	Traces all EGP packets sent and received
update	Logs all routing updates.

The autonomous system stanza specifies the autonomous system number. This must be specified if running EGP.

```
traceflags      internal external route egp update
autonomoussystem        178
```

The following **egpneighbor** stanza specifies with whom you are going to perform EGP with. This line says that your EGP neighbor is the host `192.100.9.1`. The **defaultegpmetric** stanza specifies that when there are no routing restrictions, the default EGP metric is `132`.

```
egpneighbor             192.100.9.1
defaultegpmetric        132
#
```

The next line indicates that for network `192.20.9` the gateway is `192.101.9.3` with a hop count of `50` using RIP protocol. This is a static route.

The **egpnetsreachable** stanza restricts EGP announcements to those networks listed:

```
net 192.200.9 gateway 192.101.9.3 metric 50 rip
egpnetsreachable 192.200.9 192.101.9
```

The following list is a list of static routes showing the host address, gateway address, hop count, and protocol used:

```
# Static routes
host 129.140.46.1       gateway 192.100.9.1     metric 5 rip
host 192.102.9.2        gateway 192.100.9.1     metric 5 rip
host 192.104.9.2        gateway 192.100.9.1     metric 5 rip
host 149.140.3.12       gateway 192.100.9.1     metric 5 rip
host 129.140.3.12       gateway 192.100.9.1     metric 5 rip
host 129.140.3.13       gateway 192.100.9.1     metric 5 rip
host 129.140.3.14       gateway 192.100.9.1     metric 5 rip
host 192.3.3.54         gateway 192.101.9.3     metric 5 rip
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/gated.conf Contains the configuration information for the **gated** daemon.

Related Information

The **kill** command.

The **gated** daemon, **routed** daemon.

Understanding the Protocols for TCP/IP, Understanding Routing for TCP/IP, Understanding Gateways for TCP/IP, and How to Configure the gated Daemon in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures.*

TCP/IP gateways File Format

Purpose

Specifies Internet routing information to the **routed** daemon on a network.

Description

The **/etc/gateways** file identifies gateways for the **routed** daemon. Ordinarily, the daemon queries the network and builds routing tables. The daemon builds the tables from routing information transmitted by other hosts directly connected to the network. However, there may be gateways that the daemon cannot identify through its queries. These unidentified gateways are known as *distant gateways*. Such gateways should be identified in the **gateways** file which the **routed** daemon reads when it starts.

The general format of an entry (contained on a single line) in the **gateways** file is:

```
Destination Name1
gateway Name2 metric
Value Type
```

Following is a brief description of each element in an **gateways** file entry:

Destination A keyword that indicates whether the route is to a network or to a specific host. The two possible keywords are **net** and **host**.

Name1 The name associated with *Destination*. The *Name1* variable can be either a symbolic name (as used in the **/etc/hosts** or **/etc/networks** file) or an Internet address specified in dotted decimal format.

gateway An indicator that the following string identifies the gateway host.

Name2 The name or address of the gateway host to which messages should be forwarded.

metric An indicator that the next string represents the hop count to the destination host or network.

Value The hop count, or number of gateways from the local network to the destination network.

Type A keyword that indicates whether the gateway should be treated as active, passive, or external. The three possible keywords are:

active An active gateway is treated like a network interface. That is, it is expected to exchange RIP routing information. Information about it is maintained in the internal routing tables as long as it is active and is included in any routing information that is transmitted through RIP. If it does not respond for a period of time, the route associated with it is deleted from the internal routing tables.

passive A passive gateway is not expected to exchange RIP routing information. Information about it is maintained in the routing tables indefinitely and is included in any routing information that is transmitted through RIP.

external An external gateway is identified to inform the **routed** daemon that another routing process will install such a route and that alternative routes to that destination should not be installed. Information about external gateways is not maintained in the internal routing tables and is not transmitted through RIP.

Note: These routes must be to networks.

Examples

1. To specify a route to a network through a gateway host with an entry in the **gateways** file, enter a line in the following format:

   ```
   net net2 gateway host4 metric 4 passive
   ```

   This example specifies a route to a network, `net2`, through the gateway `host4`. The hop count `metric` to `net2` is `4` and the gateway is treated as `passive`.

2. To specify a route to a host through a gateway host with an entry in the **gateways** file, enter a line in the following format:

   ```
   host host2 gateway host4 metric 4 passive
   ```

   This example specifies a route to a host, `host2`, through the gateway `host4`. The hop count `metric` to `host2` is `4` and the gateway is treated as `passive`.

3. To specify a route to a host through an active Internet gateway with an entry in the **gateways** file, enter a line in the following format:

   ```
   host host10 gateway 192.100.11.5 metric 9 active
   ```

   This example specifies a route to a specific host, `host10`, through the gateway `192.100.11.5`. The hop count `metric` to `host10` is `9` and the gateway is treated as `active`

4. To specify a route to a host through a passive Internet gateway with an entry in the **gateways** file, enter a line in the following format:

   ```
   host host10 gateway 192.100.11.5 metric 9 passive
   ```

5. To specify a route to a network through an external gateway with an entry in the **gateways** file, enter a line in the following format:

   ```
   net net5 gateway host7 metric 11 external
   ```

   This example specifies a route to a network, `net5`, through the gateway `host7`. The hop count `metric` to `net5` is `11` and the gateway is treated as `external` (that is, it is not advertised through RIP but is advertised through an unspecified routing protocol).

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/etc/gateways Contains the Internet routing information for the **routed** command.

/usr/lpp/tcpip/samples/gateways
Contains the sample **gateways** file. This file also contains directions for its use.

Related Information

The **routed** daemon.

Understanding Gateways for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures.*

TCP/IP hosts File Format

Purpose

Defines the Internet Protocol (IP) name and address of the local host and specifies the names and addresses of remote hosts.

Description

The **/etc/hosts** file contains the Internet Protocol (IP) host names and addresses for the local host and other hosts in the Internet network. This file is used to resolve a name into an address (that is, to translate a host name into its Internet address). When your system is using a name server, the file is accessed only if the name server cannot resolve the host name.

When the local host is using the DOMAIN protocol, the resolver routines query a remote DOMAIN name server before searching this file. In a flat network with no name server, the resolver routines search this file for host name and address data.

Entries in the **hosts** file have the following format:

```
Address HostName HostName HostName HostName
```

In this entry, *Address* is an IP address specified in either dotted decimal or octal format, and *HostName* is the name of a host specified in either relative or absolute domain name format. If you specify the absolute domain name, the portion of the name preceding the first . (dot) has a maximum length of 63 characters and cannot contain blanks. For both formats of the name, the total number of characters cannot exceed 255 characters, and each entry must be contained on one line. Multiple *HostNames* (or aliases) can be specified.

This file can contain two special case entries that define reserved (or well–known) host names. These host names are:

timeserver Identifies a remote time server host. This host name is used by the **setclock** command.

printserver Identifies the default host for receiving print requests.

In this **hosts** file entry, the *Address* parameter is an IP address specified in either dotted decimal or octal format, and each *HostName* parameter is a host name specified in either relative or absolute domain name format. These never have the full domain name listed; they are always listed as either `printserver` or `timeserver`.

Note: The local **/etc/resolv.conf** file defines where DOMAIN name servers are, and the name server file defines where Internet services are available. Although it is not necessary to define well–known hosts in the **hosts** file when using the DOMAIN protocol, it may be useful if they are not defined by your name server.

Entries in this file can be made using the System Management Interface Tool (SMIT), by using the **hostent** command, or by creating and editing the file with an editor.

Examples

In these examples, the name of the local host is the first line in each **hosts** file. This is to help you identify the host whose file is being displayed. Your host does not have to be defined on the first line of your **hosts** file.

1. The following sample entries might be contained in the **hosts** files for two different hosts on a network that is not running a DOMAIN name server:

   Host1

   ```
   185.300.10.1  host1
   185.300.10.2  host2
   185.300.10.3  host3
   185.300.10.4  host4 merlin
   185.300.10.5  host5 arthur king
   185.300.10.5   timeserver
   ```

   Host 2

   ```
   185.300.10.2  host2
   185.300.10.1  host1
   185.300.10.3  host3
   185.300.10.4  host4 merlin
   185.300.10.5  host5 arthur king
   ```

   In this sample network with no name server, the **hosts** file for each host must contain the Internet address and host name for each host on the network. Any host that is not listed cannot be accessed. The host at Internet address `185.300.10.4` in this example can be accessed by either name: `host4` or `merlin`. The host at Internet address `185.300.10.5` can be accessed by any of the names `host5`, `arthur`, or `king`.

2. Following is a sample entry in the **hosts** files for a different host on a DOMAIN network, but the host is not the name server, and the host is keeping some additional host names for a smaller network:

   Host 5

   ```
   128.114.1.15  name1.xyz.aus.ibm.com  name1
   128.114.1.14  name2.xyz.aus.ibm.com  name2
   128.114.1.16 name3.xyz.aus.ibm.com  name3
   ```

   In this sample, `host5` is not a name server, but is attached to a DOMAIN network. The **hosts** file for `host5` contains address entries for all hosts in the smaller network, and the DOMAIN data files contain the DOMAIN database. The entries in the `host5` **hosts** file that begin with `128.114` indicate that `host5` resolves names for hosts on the smaller network.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/hosts	Contains the host names and addresses.

Related Information

The **hostent** command, **setclock** command.

The **gethostbyaddr** routine

Understanding the DOMAIN Protocol, Understanding the Internet Protocol (IP), Understanding Naming for TCP/IP, Understanding the SMIT Interface for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP hosts.equiv File Format

Purpose

Specifies remote systems that can execute commands on the local system.

Description

The **/etc/hosts.equiv** file defines which remote hosts (computers on a network) are permitted to execute certain commands on the local host without supplying a password. The remote host must have a user name that is the same as the user name on the local host. The user name cannot be `root`. The format of the **hosts.equiv** file is a simple list of host names.

The **lpd**, **rlogind**, and **rshd** daemons check for the existence of a **hosts.equiv** file.

If you are using a DOMAIN name server and want to grant access to hosts in a different domain, you must specify the full domain name of each host.

Entries in this file can be made using the System Management Interface Tool (SMIT) or using the **ruser** command.

Example

Following are sample entries in an **hosts.equiv** file:

```
host1
host2
host3
host4
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/hosts.equiv	Contains the list of equivalent hosts.

Related Information

The **lpd** daemon, **rlogind** daemon, **rshd** daemon, **ruser** command.

Understanding Naming for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP hosts.lpd File Format

Purpose

Specifies remote hosts that can print on the local host.

Description

The **/etc/hosts.lpd** file defines which remote systems are permitted to print on the local system. The remote systems listed in this file are not given the full privileges given to files listed in the **/etc/hosts.equiv** file. The format of the **hosts.lpd** file is a simple list of host names.

Entries in this file can be made using the System Management Interface Tool (SMIT) or by using the **ruser** command.

Example

Following are example entries in an **hosts.lpd** file:

```
host12
host23
host25
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/hosts.lpd	Specifies foreign hosts that can print on the local host.

Related Information

The **lpd** daemon, **ruser** command.

The **/etc/hosts.equiv** file.

TCP/IP inetd.conf File Format

Purpose

Defines how the **inetd** daemon handles Internet service requests.

Description

In previous releases of AIX TCP/IP, this file was the default configuration file for the **inetd** daemon. The **inetd** daemon now reads the **InetServ** database (stored in the ODM) by default. You can still use the **/etc/inetd.conf** file format as the configuration file for the **inetd** daemon. However, each time you modify the file format, you must run the **inetimp** command to keep **inetd.conf** in sync with the **InetServ** database information. If you modify the **InetServ** database information (using SMIT or an ODM tool), the **inetexp** command is run automatically.

The following daemons are controlled by the **inetd** daemon:

- **comsat**
- **ftpd**
- **telnetd**
- **rshd**
- **rlogind**
- **rexecd**
- **fingerd**
- **tftpd**
- **talkd**
- **uucpd**

Service Requests

The Internet service requests that are supported internally by the **inetd** daemon are generally used for debugging. They include the following internal services:

ECHO	Returns data packets to a client host.
DISCARD	Discards received data packets.
CHARGEN	Discards received data packets and sends predefined or random data.
DAYTIME	Sends the current date and time in human–readable form.
TIME	Sends the current date and time in machine–readable form.

The **inetd** daemon reads its configuration file only when the **inetd** daemon starts, when the **inetd** daemon receives a **SIGHUP** signal, or when the **src refresh –s inetd** command is entered. Each line in the **inetd** configuration file defines how to handle one Internet service request.

Each line is of the form:

```
ServiceName SocketType ProtocolName
Wait/NoWait UserName ServerPath
ServerArgs
```

These fields must be separated by spaces or tabs. The fields have the following meanings:

ServiceName Contains the name of an Internet service defined in the **/etc/services** file. For services provided internally by the **inetd** daemon, this name must be the official name of the service. That is, the name must be identical to the first entry on the line that describes the service in the **/etc/services** file.

SocketType Contains the name for the type of socket used for the service. You can use either the **stream** value for a stream socket or the **dgram** value for a datagram socket.

ProtocolName Contains the name of an Internet protocol defined in the **/etc/protocols** file. For example, use the **tcp** value for a service that uses the TCP/IP protocol and the **udp** value for a service that uses the UDP protocol.

Wait/NoWait Contains either the **wait** or the **nowait** instruction for datagram sockets and the **nowait** instruction for stream sockets. The *Wait/NoWait* field determines whether the **inetd** daemon waits for a datagram server to release the socket before continuing to listen at the socket.

UserName Specifies the user name that the **inetd** daemon should use to start the server. This variable allows a server to be given less permission than the root user.

ServerPath Specifies the full path name of the server that the **inetd** daemon should execute to provide the service. For services that the **inetd** daemon provides internally, this field should be internal.

ServerArgs Specifies the command line arguments that the **inetd** daemon should use to execute the server. These arguments begin with the name of the server used. For services that the **inetd** daemon provides internally, this field should be empty.

Example

The following are example entries in the **/etc/inetd.conf** file for an **inetd** daemon that:

- Uses the **ftpd** daemon for servicing **ftp** requests
- Uses **talkd** daemon for **ntalk** requests
- Provides time requests internally.

```
ftp stream tcp nowait root /etc/ftpd ftpd
ntalk dgram udp wait root /etc/talkd talkd
time stream tcp nowait root internal
time dgram udp wait  root internal
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/inetd.conf Contains the configuration information for the **inetd** daemon.

Related Information

The **kill** command.

The **inetd** daemon.

The **inetimp** command, **inetexp** command.

How to Configure the inetd Daemon, Understanding the Transmission Control Protocol (TCP), Understanding the User Datagram Protocol (UDP, Understanding TCP/IP Daemons, and Understanding SRC Control of TCP/IP Daemons in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

The ODM Overview in *General Programming Concepts*.

TCP/IP named.boot File Format

Purpose

Defines how **named** initializes the DOMAIN name server file.

Description

The **/etc/named.boot** file is the default configuration (or boot) file for the **named** server. If the **named** daemon is started without specifying an alternate file, the **named** daemon reads this file for information on how to set up the local name server file. This file is directly accessed by local kernel resolver routines on the name server host and is indirectly accessed through **named** service requests by remote hosts.

Note: The **named** daemon reads the startup file only when the **named** daemon starts or when the **named** daemon receives an SRC **refresh** command or a SIGHUP signal.

The records in the **named.boot** file tell the **named** daemon what type of server it is, which domains it has authority over (its zones of authority), and where to get the data for initially setting up its name resolution database.

Any data files referenced in the **named** startup file must be in Standard Resource Record Format. These data files can have any name. However, for convenience in maintaining the **named** database, they are generally given names in the following form: **/etc/named.***extension*. The general format of **named** data files is described in DOMAIN Data File, DOMAIN Reverse Data File, DOMAIN Cache File, and DOMAIN Local File.

Comments in the boot file begin with a ; (semicolon) and end at the end of the line.

The **named.boot** file is organized in lines; valid lines and their formats are:

cache *File Name*

Indicates that the local **named** server is a caching name server for the domain specified in the *Domain* field and that the **named** daemon is to get the data describing the domain from the file specified in the *Source* field.

The name server first needs to know the *root* name server (the name server on the network with the highest authority). The root name server is established in the **named.boot** file by specifying the root server file name (**named.ca**) as the cache for this name server.

Note: The **named** daemon does not provide other hosts with the information contained in a cache file. Cache files are usually used for listing the name servers for domains higher than the local domain.

An example of the **cache** line follows:

```
cache            .            /etc/named.ca
```

directory *Path* Causes the server to change its working directory to the directory specified. This can be important for the correct processing of $INCLUDE files in primary zone files as well as in locating data files for the name server. An example of the **directory** line follows:

```
directory /usr/local/domain
```

Note: If no directory line is specified here, the full path name must be specified in all other lines specifying files.

domain *Domain*

Indicates that the following *Domain* entry is the name of the default domain for the name server. When the **named** daemon receives a query with a domain or host name that does not end with a . (period), the **named** daemon appends the default domain name to the queried name. An example of the **domain** line follows:

```
domain          abc.aus.ibm.com
```

forwarders *IPAddresses*

The forwarders line specifies the addresses of site–wide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarders, then the server will send all queries for data not in the cache to the forwarders first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, the server continues as it would have without the forwarders line unless it is in slave mode. The forwarding facility is useful to cause a large site–wide cache to be generated on a master, and to reduce traffic over links to outside servers. It can also be used to allow servers to run that do not have access directly to the Internet, but wish to act as though they do. An example of a forwarders line follows:

```
forwarders      10.0.0.78 10.2.0.78
```

primary *Domain File Name*

Indicates that the local **named** server is the primary name server for the domain specified in the *Domain* field and that the **named** daemon is to get the data describing the domain from the file specified in the *File Name* field. An example of the **primary** line follows:

```
primary abc.aus.ibm.com    /etc/named.abcdata
```

secondary *Domain IPAddresses File Name*

Indicates that the local **named** server is a secondary name server for the domain specified in the *Domain* field and that the **named** daemon is to get the data describing the domain from one or more remote primary name servers using the Internet address or addresses specified in the *IPAddresses* field. The **named** daemon tries each address in the order listed until it successfully receives the data from one of the name servers. The **named** daemon will backup the information it receives from the primary name server in the file specified in the *File Name* field. Whenever a new copy of the domain information is received from one of the primary servers, this file will be updated. The daemon will use this file as its initial cache any time the primary name server is down. In the secondary line, the *File Name* field is required. Examples of the secondary line follow:

```
secondary abc.aus.ibm.com 192.9.20.1 192.9.20.2\
/etc/named.abc.bak
secondary 201.9.192.in–addr.arpa 192.9.20.1 192.9.20.2\
/etc/named.abc.bak
```

slave The slave line is used to put the server in slave mode. In this mode, the server only makes queries to forwarders. This option is normally used on machines that wish to run a server but for physical or administrative reasons cannot be given access to the Internet, but have access to a host that does have access to the Internet. The format of the slave line follows:

```
slave
```

sortlist Indicates networks that take precedence over other networks. Requests for name resolution from a host on the same network as the server receive local network addresses listed first, addresses on the sortlist listed second, and all other addresses listed last. The sortlist line is only acted upon at initial startup. When reloading the name server with a **SIGHUP** signal, this line will be ignored.

```
sortlist 192.9.200.14 129.35.17.2
```

Examples

The following examples show the various ways to use the **named** boot file. In these examples, two networks are represented: `abc` and `xyz`.

Network `abc` consists of:

- `gobi.abc`, the primary name server for the `abc` network, 192.9.201.2
- `mojave.abc`, a host machine, 192.9.201.6
- `sandy.abc`, secondary name server for the `abc` network and gateway between `abc` and `xyz`, 192.9.201.3.

Network `xyz` consists of:

- `kalahari.xyz`, primary name server for the `xyz` network, 160.9.201.4
- `lopnor.xyz`, a host machine, 160.9.201.5
- `sahara.xyz`, a host machine and cache–only name server for the `xyz` network, 160.9.201.13
- `sandy.xyz`, a secondary name server for the `xyz` network and gateway between `abc` and `xyz`, 160.9.201.3

Note: Note that `sandy`, a gateway host, is on both networks and also serves as secondary name server for both.

1. The **/etc/named.boot** file for `gobi.abc`, primary name server for network `abc`, contains these entries:

```
;
;boot file for abc primary server  – gobi.abc
;type           domain                  source file or host
;
domain          abc
primary         abc                     /etc/named.abcdata
primary         201.9.192.inn–addr.arpa /etc/named.abcrev
primary         0.0.127.in–addr.arpa    /etc/named.abclocal
```

2. The **/etc/named.boot** file for `kalahari.xyz`, primary name server for network `xyz`, contains these entries:

```
;boot file for abc primary server  -  kalahari.xyz
;
;type        domain                    source file or host
;
domain          xyz
primary         xyz                      /etc/named.xyzdata
primary         9.160.in-addr.arpa       /etc/named.xyzrev
primary         0.0.127.in-addr.arpa     /etc/named.xyz.local
```

3. The **/etc/named.boot** file for `sandy`, secondary name server for networks `abc` and `xyz`, contains the following entries:

```
;boot file for secondary server for abc and xyz - sandy
;
;type      domain                  source file or host
;
domain       abc
directory /etc
secondary  abc                     192.9.201.2  named.abcdata.bak
secondary  xyz                     160.9.201.4  named.xyzdata.bak
secondary  201.9.192.in-addr.arpa  192.9.201.2  named.abcrev.bak
secondary  9.160.in-addr.arpa      192.9.201.4  named.xyzrev.bak
primary    0.0.127.in-addr.arpa                 named.seclocal
```

4. The **/etc/named.boot** file for `sahara`, a cache-only name server for the network `xyz`, contains the following entries:

```
;boot file for cache-only server for xyz - sahara
;
;type           domain                  source file or host
;
domain          xyz
cache           .                       /etc/named.ca
primary         0.0.127.in-addr.arpa    /etc/named.calocal
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/etc/named.boot Contains the configuration information for the **named** server.

/usr/lpp/tcpip/samples/named.boot
Contains the sample **named.boot** file. This file also contains directions for its use.

Related Information

The **named** daemon.

The TCP/IP DOMAIN cache file, TCP/IP DOMAIN local file, TCP/IP DOMAIN data file, TCP/IP DOMAIN Reverse data file, TCP/IP **rc.tcpip** file.

Configuring Name Servers for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP .netrc File Format

Purpose

Specifies automatic login information for the **ftp** and **rexec** commands.

Description

The **.netrc** file contains the information used by the automatic login feature of the **rexec** and **ftp** commands. It is a hidden file in a user's home directory and must be owned either by the user executing the command or by the root user. If the **.netrc** file contains a login password, the file's permissions must be set to 600 (read and write by owner only).

Note: The **.netrc** file is not used by any programs when the **secure.tcpip** command is running on your system.

The **.netrc** can contain the following entries (separated by blanks, tabs, or new lines):

machine *HostName*

The *HostName* variable is the name of a remote host. This entry begins the definition of the autologin process for the specified host. All following entries up to the next machine entry or the end of the file apply to that host.

login *UserName*

The *UserName* variable is the full domain user name for use at the remote host. If this entry is found, the autologin process initiates a login using the specified name. If this entry is missing, the autologin process fails.

password *Password*

The *Password* variable is the login password to be used. The autologin process supplies this password to the remote server. A login password must be established at the remote host and that password must be entered in this file, or the autologin process fails and the user is prompted for the login password.

account *Password*

The *Password* variable is the account password to be used. If this entry is found and an account password is required at the remote host, the autologin process supplies the password to the remote server. If the remote host requires an account password but this entry is missing, the autologin process prompts for the account password.

macdef *MacroName*

The *MacroName* variable is the name of an **ftp** subcommand macro. The macro is defined to contain all of the following **ftp** subcommands up to the next blank line or the end of the file. If the macro is named **init**, the **ftp** command executes the macro upon successful completion of the autologin process. The **rexec** command does not recognize a **macdef** entry.

Example

To use the example **.netrc** file to create your own, follow these steps:

1. Copy the **/usr/lpp/tcpip/samples/netrc** file to your home directory.
2. Edit the **$HOME/netrc** file to supply the appropriate *HostName*, *UserName*, and *Password* variables.
3. Set the permissions on the **$HOME/netrc** file to 600.
4. Rename the **.netrc** file (the initial . (dot) causes the file to be hidden).
5. The following is an example of an entry in a **.netrc** file.

```
machine host1.austin.ibm.com login fred password bluebonnet
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

$HOME/.netrc	Contains automatic login information.
/usr/lpp/tcpip/samples/netrc	Contains a sample **.netrc** file with directions for its use.

Related Information

The **ftp** command, **rexec** command.

TCP/IP networks File Format

Purpose

Contains the network name information.

Description

The **/etc/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 the **networks** file is:

```
Name Number Aliases
```

The fields are described as follows:

Name Official network name

Number Network number

Aliases Unofficial names used for the network.

Items on a line are separated by one or more blanks or tab characters. Comments begin with a # (pound sign). Routines that search the **networks** file do not interpret characters from the beginning of a comment to the end of that line. Network numbers are specified in dotted decimal 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 database maintained at the Network Information Center (NIC). The file may need to be modified locally to include unofficial aliases or unknown networks.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/etc/networks	Contains the network name database.
/usr/lpp/tcpip/samples/networks	Contains a sample **networks** file. This file also contains directions for its use.

Related Information

The **routed** daemon.

The **getnetent** routines.

TCP/IP protocols File Format

Purpose

Defines the Internet protocols used on the local host.

Description

The **/etc/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. Each entry is of the form:

Name Number Aliases

The fields contain the following information:

Name Official Internet Protocol name

Number Protocol number

Aliases Unofficial names used for the protocol.

Items on a line are separated by one or more blanks or tab characters. Comments begin with the # (pound sign), and routines that search the **protocols** file do not interpret characters from the beginning of a comment to the end of the line. A protocol name can contain any printable character except a field delimiter, new line character, or comment character.

The lines appear as follows:

```
ip              0       #dummy for IP
icmp            1       #control message protocol
#ggp            2       #gateway^2 (not normally used)
tcp             6       #tcp
#egp            8       #exterior gateway protocol
#pup            12      #pup
udp             17      #user datagram protocol
#idp            22      #xns idp
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/protocols Defines Internet protocols for local host.

Related Information

The **getprotoent** routines.

Understanding Protocols for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter of *Communication Concepts and Procedures.*

TCP/IP rc.net File Format

Purpose

Defines host configuration for the following areas: network interfaces, host name, default gateway, and any static routes.

Description

The **/etc/rc.net** file is a shell script that contains configuration information. The stanzas allow you to enable the network interfaces and set the host name, the default gateway, and any static routes for the current host. This file can be used as a one-step configuration alternative to using individually the set of commands and files necessary to configure a host.

The **rc.net** shell script is run by the configuration manager program during the second phase of configuration. If TCP/IP is installed, a second script, **rc.tcpip**, is run from **init** after the second phase of configuration has completed and after **init** has started the SRC master. Stanzas in the file should appear in the order in which they are presented here.

Using the Configuration Methods

These stanzas use the configuration methods for TCP/IP to manipulate the ODM database.

Configuring Network Interfaces

For each network adapter that has been previously configured, a set of stanzas is required. The following stanzas define, load, and configure the appropriate network interfaces for every configured network adapter. These configuration methods require that the interface and protocol information be entered in the ODM database using either SMIT or the high level configuration commands (i.e., **mkdev**). The network interface configuration information is held in the running system only, and must be reset at each system restart.

```
/etc/network/defif          >>$LOGFILE   2>&1

/etc/network/cfgif    $*    >>$LOGFILE   2>&1
```

The **defif** method defines the network interfaces. The **cfgif** method configures the network interfaces in the configuration database. The second part of the stanzas indicates that output should be sent to a log file. The log file must include the full path name. If no log file is specified, the default log file is **/dev/null**.

Along with the network interface configuration, additional commands must be executed for X.25 and SLIP interfaces: the **x25xlate** command for X.25 interfaces and the **slattach** command for SLIP connections.,The **x25xlate** command loads the X.25 translation table into the kernel and the **slattach** command is used to assign a tty line to an interface for SLIP. For each SLIP interface, the **slattach** command must be executed for the appropriate tty.

Setting the Host Name, Default Gateway, and Any Static Routes

The following stanzas set the host name, default gateway, and static routes using the **definet** subroutine and the **cfginet** subroutine to alter the ODM database for the inet0 object.

```
/etc/definet   >>$LOGFILE 2>&1
/etc/cfginet   >>$LOGFILE 2>&1
```

The second part of the stanzas indicates that output should be sent to a log file. The log file must include the full path name. If no log file is specified, the default log file is **/dev/null**.

Using Traditional Configuration Commands

These stanzas use configuration commands for TCP/IP to set configuration values.

Configuring Network Interfaces

The following stanza defines, loads, and configures the specified network interface.

```
/etc/ifconfig Interface inet InternetAddress up  >>$LOGFILE 2>&1
```

The *Interface* parameter should specify the type and number of the interface (i.e., tr0). The *InternetAddress* parameter should specify the Internet address of the interface (i.e., 192.1.8.0).

The last part of the stanza indicates that output should be sent to a log file. The log file must include the full path name. If no log file is specified, the default log file is **/dev/null**.

Setting the Host Name, Default Gateway, and Any Static Routes

These stanzas should follow any stanzas for the network interfaces. These stanzas use the **hostname** command to set the host name and the **route** command to define the default gateway and any static routes. The static route information is held in the running system only, and must be reset at each system restart.

```
/etc/hostname  Hostname       >>$LOGFILE 2>&1

/etc/route  add  0  Gateway  >>$LOGFILE 2>&1

/etc/route  add  DestinationAddress  Gateway  >>$LOGFILE 2>&1
```

The **add** parameter for the **route** command adds a static route to the host. This route can be to the default gateway (by specifying a hop count, or metric, of 0), or to another host through a gateway.

The last part of the stanzas indicates that output should be sent to a log file. The log file must include the full path name. If no log file is specified, the default log file is **/dev/null**.

Miscellaneous Functions

Use these stanzas to set the host id and user name. By default, the host id and user name are set to the host name. However, these stanzas can be altered to customize the host id and user name.

```
/usr/bin/hostid 'hostname'

/bin/uname -s 'hostname | sed -e 's/\..*$//''  >>$LOGFILE 2>&1
```

To customize these stanzas, replace the `hostname` entry in single quotation marks with the desired host id or user name.

The second part of the user name stanza indicates that output should be sent to a log file. The log file must include the full path name. If no log file is specified, the default log file is **/dev/null**.

Load Network File System (NFS)

If you have the Network File System (NFS) installed on the current host, the following stanza will load and configure the NFS kernel extension:

```
if [ -x /etc/gfsinstall -a -x /etc/nfs.ext ] ; then
        /etc/gfsinstall -a /etc/nfs.ext >>$LOGFILE 2>&1
```

The last part of the NFS stanza indicates that output should be sent to a log file. The log file must include the full path name. If no log file is specified, the default log file is **/dev/null**.

Examples

1. To set up a Token-Ring interface using the **ifconfig** command, include the following stanza:

   ```
   /etc/ifconfig tr0 inet 192.1.8.0 up >>$LOGFILE 2>&1
   ```

   This stanza will set up the Token-Ring interface, unit zero, with the Internet address 192.1.8.0.

2. To set the host name using the **hostname** command, include the following stanza:

   ```
   /etc/hostname robo.austin.ibm.com   >>$LOGFILE 2>&1
   ```

   The stanza will set the host name as robo.austin.ibm.com. The host name in this example includes domain and subdomain information, which is necessary if the host is using the domain naming system.

3. To set up a default gateway using the **route** command, include the following stanza:

   ```
   /etc/route add 0 192.100.13.7    >>$LOGFILE 2>&1
   ```

   The value `0` for the *Metric* parameter means that any packets sent to destinations not previously defined and not on a directly connected network go through the default gateway. The `192.100.13.7` address is that of the gateway chosen to be the default.

4. To set up a static route using the **route** command, include the following stanza:

   ```
   /etc/route add net 192.100.201.7 192.100.13.7 >>$LOGFILE 2>&1
   ```

   The `192.100.201.7` address is that of the receiving computer (the *Destination* parameter). The `192.100.13.7` address is that of the routing computer (the *Gateway* parameter).

Implementation Specifics

This file is part of TCP/IP Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/rc.tcpip Initializes daemons each system IPL.

Related Information

The **defif** subroutine, **cfgif** subroutine, **definet** subroutine, **cfginet** subroutine.

The **ifconfig** command, **init** command, **x25xlate** command, **slattach** command, **hostname** command, **route** command.

The **rc.tcpip** file, **inetd** daemon, **sendmail** daemon.

TCP/IP resolv.conf File Format

Purpose

Defines DOMAIN name-server information for local resolver routines.

Description

If the **/etc/resolv.conf** file exists, the local resolver routines either use a local name resolution database maintained by a local **named** daemon (a process) to resolve Internet names and addresses, or they use the Domain Name Protocol (DOMAIN protocol) to request name resolution services from a remote DOMAIN name server host. In either case, if the name resolution information is unavailable, the routines then attempt to use the **/etc/hosts** file for name resolution.

Note: If the **resolv.conf** file does not exist, the resolver routines attempt name resolution using the local **/etc/hosts** file.

On a host that is not a name server but is to use a name server to obtain name resolution information, the **resolv.conf** file should contain the Internet address of the name server. If the host is a name server, the **resolv.conf** file *must* exist and should have a length of 0 (zero).

The **resolv.conf** file contains at most one domain entry and 1 to 16 name server entries.

A domain entry tells the resolver routines which default domain name to append to names that do not end with a . (dot). There can be only one domain entry. This entry is of the form:

`domain` *`DomainName`*

The *DomainName* variable is the name of the local Internet domain. If there is no domain entry in the file, the default domain is the domain returned by the **gethostbyname** subroutine (that is, everything following the first period). If the host name does not have a domain name included, the root (.) domain is assumed.

A name server entry tells the resolver routines the Internet address of a remote DOMAIN name server for the local domain. This entry is of the form:

`nameserver` *`Address`*

The *Address* variable is the dotted decimal address of the remote name server. If more than one name server is listed, the resolver routines query each name server (in the order listed) until either the query succeeds or the maximum number of attempts have been made.

Each line in the **resolv.conf** file must begin with either `domain` or `nameserver`, followed by blanks or tabs and a corresponding *DomainName* or *Address*.

Entries in this file can be made using the System Management Interface Tool (SMIT), by using the **namerslv** command, or by creating and editing the file with an editor.

Example

To define a domain host that is not a name server, enter:

```
domain abc.aus.ibm.com
nameserver 192.9.201.1
nameserver 192.9.201.2
```

The example contains entries in the **resolv.conf** file for a host that is not a name server.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Files

/etc/resolv.conf	Defines name server information for resolver routines.
/usr/lpp/tcpip/samples/resolv.conf	Contains the sample **resolv.conf** file.

Related Information

The **named** daemon.

The **namerslv** command.

The **/etc/hosts** file.

The **gethostbyaddr** subroutine, **gethostname** subroutine.

Understanding Naming for TCP/IP and Configuring Name Servers for TCP/IP in the Transmission Control Protocol chapter of *Communication Concepts and Procedures*.

TCP/IP .rhosts File Format

Purpose

Specifies remote users that can use a local user account on a network.

Description

The **$HOME/.rhosts** file contains a list of remote users who are not required to supply a login password when they execute the **rcp**, **rlogin** and **rsh** commands using a local user account. This file is a hidden file in the local user's home directory and must be owned by the local user. The permissions of the **.rhosts** file should be set to 600 (read and write by owner only). Each entry in the file is of the form:

Host UserName

The *Host* variable is the name of the remote host, and the *UserName* variable is the login name of the remote user. If the remote host is in a different domain from the local host, the full domain name must be specified.

Example

To allow remote users to log in to someone else's home directory, enter:

```
venus geo
venus mark
```

The example shows entries in an **.rhosts** file on host `zeus`. For example the file may be the **/u/amy/.rhosts** file. These entries allow users `mark` and `geo` at remote host `venus` to log in to user amy's home directory on host `zeus`.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

$HOME/.rhosts	Specifies remote users that can use a local user account.

Related Information

The **rcp** command, **rlogin** command, **rsh, remsh** command.

TCP/IP services File Format

Purpose

Defines the sockets and protocols used for Internet services.

Description

The **/etc/services** file contains information about the known services used in the DARPA Internet network. Each service is listed in this file on a single line of the form:

ServiceName PortNumber/ProtocolName Aliases

The fields contain the following information:

ServiceName	Official Internet service name
PortNumber	Socket port number used for the service
ProtocolName	Transport protocol used for the service
Aliases	List of unofficial service names.

Items on a line are separated by blanks or tabs. Comments begin with a # (pound sign) and continue to the end of the line.

Like the **/etc/inetd.conf** file, any time the **services** file is changed, the **inetimp** command must be called to create or update the Object Data Manager (ODM) object class **InetServ**. Anytime the Object Data Manager (ODM) **InetServ** object class is updated, the **inetexp** command must be called to create or update the contents of the **services** and **inetd.conf** files to make sure that their contents remain the same. It is important to make sure their contents always reflect the same information because System Resource Controller (SRC) and the **inetd** daemon use the information from the ODM object class. When the ODM object is updated by the **inetserv** command or through SMIT, the **inetexp** export routine is automatically called.

Example

Entries in the **services** file for the **inetd** internal services might look like this:

```
echo          7/tcp
echo          7/udp
discard       9/tcp       sink null
discard       9/udp       sink null
daytime      13/tcp
daytime      13/udp
chargen      19/tcp       ttytst source
chargen      19/tcp       ttytst source
ftp          21/tcp
time         37/tcp       timeserver
time         37/udp       timeserver
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

File

/etc/services	Defines sockets and protocols used for Internet services.

Related Information

The **getservent**... commands.

The **inetimp** command, **inetexp** command.

The **/etc/inetd.conf** file.

The ODM Overview in *General Programming Concepts*.

TCP/IP Standard Resource Record Format

Purpose

Defines the format of lines in the **named** data files.

Description

Records in the **named** files are called *resource records*. Files using the Standard Resource Record Format are:

- DOMAIN data file
- DOMAIN reverse data file
- DOMAIN cache file
- DOMAIN local file.

Resource records in the **named** files have the following general format:

{name} *{ttl}* *addr-class* *Record Type* *Record-Specific Data*

Field Definitions

name Varies depending on the *record type* field. The name field can specify the name of a domain, a zone of authority, the name of a host, the alias of a host or of a mailbox, or a user's login ID. The name field must begin in column one. If this field is left blank, the name defaults to the value of the previous resource record.

ttl Time to live. This specifies how long the record is stored in the database. If this field is left blank, the time to live defaults to the time to live specified in the Start of Authority record. This field is optional.

addr-class Address class of the record. There are two valid entries for this field: IN for Internet and ANY for all address classes.

Record Type The type of resource record. Valid record types are:

SOA	Start of authority record
NS	Name Server record
A	Address record
HINFO	Host Information record
WKS	Well-Known Services record
CNAME	Canonical Name record
PTR	Domain Name Pointer record
MB	Mailbox record
MR	Mail Rename Name record
MINFO	Mailbox Information record
MG	Mail Group Member record
MX	Mail Exchanger record.

Record types are described in Resource Record Types.

Record-Specific Data
These fields are dependent on the *record type* field.

Although case distinctions are kept when loading databases, all queries to the name server database are case insensitive.

Special Characters

The following characters have special meanings:

. If used in the *name* field, a free standing dot indicates the current domain.

Note: Use the dot at the end of resource records to append the path of the current domain.

.. If used in the *name* field, two free standing dots indicate the null domain name of the root domain.

@ If used in the *name* field, a free standing @ (*at* sign) indicates the current origin.

\X Where X is any character except numbers 0 through 9, a backslash preceding a character indicates that the character's special meaning should not be used. For example, `\.` can be used to put a dot or a period in the label of an entry in the name field.

\DDD Where each D is a number (0 through 9). Each number will be seen as the binary octet corresponding to the number. These octets will not be checked for special meaning.

( ) Parentheses are used to indicate that data broken into more than one line should be grouped together.

; Indicates a comment line. All characters after the semi-colon are ignored.

* An asterisk indicates wildcarding.

Special Types of Lines

There are two special types of lines that are not data lines. Instead they specify special processing. These lines are the $INCLUDE line and the $ORIGIN line.

$INCLUDE *FileName*
This line begins in column one and is followed by a file name. It indicates that the file specified should be included in the name server database. This is useful in separating different types of data into multiple files. For example:

```
$INCLUDE /usr/named/data/mailbox
```

indicates that this file should be loaded into the name server's database. Data files specified by the $INCLUDE line are not treated differently from any other **named** data file.

$ORIGIN *OriginName*
This line begins in column one and is followed by the name of a domain. This line indicates that the origin from more than one domain in a data file should be changed.

Resource Record Types

Following are descriptions and examples of the resource record types used in the **named** data files.

Start of Authority Record

Indicated by a value of SOA in the *record type* field. The Start of Authority record indicates the start of a zone of authority. There should be only one Start of Authority record per zone. However, the SOA record for the zone should be in each **named.data** and **named.rev** file on each name server in the zone. Its structure is in the following format:

```
{name}   {ttl}  addr-class   Record Type Origin                Person in Charge

@                IN          SOA       merlin.ibm.com  jane.merlin.ibm.com
         (1.1                  ;Serial
          3600                 ;Refresh
          600                  ;Retry
          3600000              ;Expire
          86400)               ;Minimum
```

Fields

name Name of the zone.

ttl Time to live.

`Serial` Version number of of this data file. This number should be incremented each time a change is made to the data. The upper limit for the number to the right of the decimal point is 9999.

`Refresh` The number of seconds after which a secondary name server checks with the primary name server to see if an update is needed. A suggested value for this field is 3600 (1 hour).

`Retry` The number of seconds after which a secondary name server is to retry after a refresh attempt fails. A suggested value for this field is 600 (10 minutes).

`Expire` The upper limit in seconds that a secondary name server is to use the data before it expires because it has not been refreshed. This value should be fairly large, and a suggested value is 3600000 (42 days).

`Minimum` The minimum time, in seconds, to use as time-to-live values in resource records. A suggested value is 86400 (one day).

addr-class Internet (IN).

Record Type Start of Authority (SOA).

Origin Name of the host on which this data file resides.

Person in Charge
Person responsible for keeping the data file current. The format is similar to a mailing address, but the @ (*at* sign) that normally separates the user from the host name is replaced by a . (period).

Name Server Record

Indicated by a value of NS in the *record type* field. The Name Server record specifies the name server responsible for a given domain. There should be one name server record for each primary server for the domain. The Name Server record can be in the **named.data** file, the **named.rev** file, the **named.ca** file, and the **named.local** file. Its structure is in the following format:

{name}	*{ttl}*	*addr-class*	*Record Type*	*Name Server's name*
		IN	NS	arthur.ibm.com

Fields

name Indicates the domain that is serviced by the specified name server. In this case, the domain will default to the value in the previous resource record.

ttl Time to live.

addr-class Internet (IN).

Record Type Name Server (NS).

Name Server's name
The name of the name server responsible for the specified domain.

Address Record

Indicated by a value of A in the *record type* field. The Address Record specifies the address for the host. Address records can be entries in the **named.ca** file, the **named.data** file, and the **named.rev** file. Its structure is in the following format:

{name}	*{ttl}*	*addr-class*	*Record Type*	*Address*
arthur		IN	A	132.10.8.1
		IN	A	10.0.4.1

Fields

name Name of the host.

ttl Time to live.

addr-class Internet (IN).

Record Type Address (A).

Address Internet address of the host in dotted decimal form. There should be one Address record for each Internet address of the host.

If the name server host for a particular domain resides inside the domain, then an A (address) resource record that specifies the address of the server is required. This address record is only needed in the server delegating the domain, not in the domain itself. If, for example, the server for domain aus.ibm.com was fran.aus.ibm.com, then the NS record and the required A record would look like:

```
aus.ibm.com.          IN      NS    fran.aus.ibm.com.
fran.aus.ibm.com.     IN      A     192.9.201.14
```

Host Information Record

Indicated by `HINFO` in the *Record Type* field. The Host Information record lists host specific information. This lists the hardware and operating system that are running at the specified host. Note that the hardware and operating system information is separated by a single space. There must be one Host Information record for each host. The HINFO record is a valid entry in the **named.data** and the **named.rev** files. Its structure is in the following format:

{*name*}	{*ttl*}	*addr-class*	*Record Type*	*Hardware OS*
		ANY	HINFO	RS/6000 AIX

Fields

name Name of the host.

ttl Time to live.

addr-class ANY. (Since host information is not specific to an address class, ANY may be used.)

Record Type Host Information(HINFO).

Hardware Make and model of hardware.

OS Name of the operating system running on the host.

Well-Known Services Record

Indicated by `WKS` in the *record type* field. The Well-Known Services lists the well-known services supported by a particular protocol at a specified address. This record is now obsolete. However, AIX TCP/IP provides the record for backward compatibility.

The services and port numbers come from the list of services in the **/etc/services** file. There should be only one WKS record per protocol per address. The WKS record is a valid entry in the **named.data** file. Its structure is in the following format:

{*name*}	{*ttl*}	*addr-class*	*Record Type*	*Address*	*Protocol*	*list of services*
		IN	WKS	125.10.0.4	UDP	(who route timed domain)
		IN	WKS	125.10.0.4	TCP	(echo telnet ftp netstat finger)

Fields

name Name of the host. In this case, the name of the host defaults to the value in the previous resource record.

addr-class Internet (IN).

ttl Time to live.

Record Type Well-known services (WKS).

Address Internet address of the adapter in dotted decimal form .

Protocol The protocol used by the list of services at the specified address.

list of services The services supported by a protocol at the specified address.

Canonical Name Record

Indicated by CNAME in the *record type* field. Specifies an alias for a canonical name. The CNAME record is the only Resource record that can use the alias of a canonical name. All other Resource records must use the full canonical (or domain) name. The CNAME record is a valid entry in the **named.data** file. For each CNAME record, there must be a corresponding Address (A) record. Its structure is in the following format:

{aliases}	*{ttl}*	*addr-class*	*Record Type*	*Canonical Name*
knight		IN	CNAME	lancelot
john		IN	CNAME	lancelot

Fields

aliases Specifies an alias by which the host is known.

ttl Time to live.

addr-class Internet (IN).

Record Type Canonical name (CNAME).

Canonical Name
Specifies the official name associated with the alias.

IN–ADDR.ARPA Record

The structure of names in the domain system is set up in a hierarchical fashion. The address of a name can be found by tracing down the domain structure, contacting a server for each label in the name. Because of this structure based on name, there is no easy way to translate a host address back into its host name.

In order to allow simple reverse translation, the IN–ADDR.ARPA domain was created. This domain uses addresses of hosts as part of a name that points to the data for that host. The IN–ADDR.ARPA domain provides an index to the resource records of each host based on its address. There are subdomains within the IN–ADDR.ARPA domain for each network, based on network number. Also, to maintain consistency and natural groupings, the 4 octets of a host number are reversed. The IN–ADDR.ARPA domain is defined by the IN–ADDR.ARPA record in the **named.boot** files and the DOMAIN hosts data file.

For example, the ARPANET is net 10, which means that there is a domain called 10.in–addr.arpa. Within this domain, there is a PTR resource record at 51.0.0.10.IN–ADDR, which points to the resource records for the host sri–nic.arpa (whose address is 10.0.0.51). Since the NIC is also on the MILNET (net 26, address 26.0.0.73), there is also a PTR resource record at 73.0.0.26.in–addr.arpa that points to the same resource records for SRI–NIC.ARPA. The format of these special pointers is defined in the following section on PTR resource records, along with the examples for the NIC.

Domain Name Pointer Record

Indicated by `PTR` in the Record Type field. This record allows special names to point to some other location in the domain. PTR resource records are mainly used in IN–ADDR.ARPA records for translation of addresses to names. PTR records should use official host names, not aliases. The PTR record is a valid entry in the **named.rev** file. Its structure is in the following format:

{aliases}	*{ttl}*	*addr-class*	*Record Type*	*Real Name*
`7.0`		`IN`	`PTR`	`arthur.ibm.com.`

Fields

aliases Specifies where in the domain this record should point. The Internet address of the host with the octets in reverse order. If you have not defined the IN–ADDR.ARPA domain in your **named.boot** file, this address must be followed by `.in–addr.arpa`.

ttl Time to live.

addr-class Internet (IN).

Record Type Pointer (PTR).

Real Name The domain name of the host to which this record points.

Gateway PTR Record

The IN–ADDR domain is also used to locate gateways on a particular network. Gateways have the same kind of PTR resource records as hosts, but they also have other PTR records used to locate them by network number alone. These records have 1, 2, or 3 octets as part of the name, depending on whether they are class A, B, or C networks, respectively.

The gateway host named `gw`, for example, connects three different networks, one each in class A, B, and C. This gateway has the standard resource records for a host in the `csl.sri.com` zone:

```
gw.csl.sri.com.         IN      A     10.2.0.2
                        IN      A     128.18.1.1
                        IN      A     192.12.33.2
```

In addition, this gateway has one of the following pairs of number–to–name translation pointers and gateway location pointers in each of the three different zones (one for each network). In each example, the number–to–name pointer is listed first, followed by the gateway location pointer.

Class A

```
2.0.2.10.in–addr.arpa.      IN      PTR     gw.csl.sri.com.
10.in–addr.arpa.            IN      PTR     gw.csl.sri.com.
```

Class B

```
1.1.18.128.in–addr.arpa.    IN      PTR     gw.csl.sri.com.
18.128.in–addr.arpa.        IN      PTR     gw.csl.sri.com.
```

Class C

```
1.33.12.192.in–addr.arpa.   IN      PTR     gw.csl.sri.com.
33.12.192.in–addr.arpa.     IN      PTR     gw.csl.sri.com.
```

For example, a user named `elizabeth` used the following resource record to have her mail delivered to host `venus.abc.aus.ibm.com.`:

```
elizabeth                   IN    MB      venus.abc.aus.ibm.com.
```

Mailbox Record

Indicated by `MB` in the *record type* field. The Mailbox record lists the machine where a user wants to receive mail. The MB record is a valid entry in the **named.data** file. Its structure is in the following format:

{aliases}	*{ttl}*	*addr-class*	*Record Type*	*Machine*
`jane`		`IN`	`MB`	`merlin.ibm.com`

Fields

aliases The user's login ID.

ttl Time to live.

addr-class Internet (IN).

Record Type Mailbox (MB).

Machine Name of the machine at which the user wants to receive mail.

Mail Rename Name Record

Indicated by `MR` in the *record type* field. The Mail Rename record allows a user to receive mail addressed to a list of aliases. The MR record is a valid entry in the **named.data** file. Its structure is in the following format:

{aliases}	*{ttl}*	*addr-class*	*Record Type*	*Corresponding MB*
`merlin`		`IN`	`MR`	`jane`

Fields

aliases Alias for the mailbox name listed in the last field.

ttl Time to live.

addr-class Internet (IN).

Record Type Mail Rename (MR).

Corresponding MB
The name of the mailbox. This record should have a corresponding MB record.

Mailbox Information Record

Indicated by `MINFO` in the *record type* field. The Mailbox Information record creates a mail group for a mailing list. This record usually has a corresponding Mail Group record, but may also be used with a Mailbox record. The MINFO record is a valid entry in the **named.data** file. Its structure is in the following format:

{name}	*{ttl}*	*addr-class*	*Record Type*	*Requests*	*Maintainer*
`postmaster`		`IN`	`MINFO`	`post-request`	`greg.ibm.com`

Fields

name The name of the mailbox.

ttl Time to live.

addr-class Internet (IN).

Record Type Mail Information record (MINFO).

Requests Where mail requests (such as a request to be added to the mailing list) should be sent.

Maintainer The mailbox that should receive error messages. This is particularly useful when mail errors should be reported to someone other than the sender.

Mail Group Member Record

Indicated by `MG` in the *record type* field. The Mail Group Member record lists the members of a mail group. The MG record is a valid entry in the **named.data** file. Its structure is in the following format:

{*mail group name*}	{*ttl*}	*addr-class*	*Record Type*	*Member Name*
`dept`		`IN`	`MG`	`Tom`

Fields

mail group name
Specifies the name of the mail group.

ttl Time to live.

addr-class Internet (IN).

Record Type Mail group member record (MG).

Member Name The login ID of the group member.

Mail Exchanger Record

Indicated by `MX` in the *record type* field. The Mail Exchanger records are used to identify machines (gateways) that know how to deliver mail to a machine that is not directly connected to the network. Wildcard names containing the character * (asterisk) may be used for mail routing with MX records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. The MX record is a valid entry in the **named.data** file. Its structure is in the following format:

{*name*}	{*ttl*}	*addr-class*	*Record Type*	*Pref value*	*mail exchanger*
`Ann.ibm.com`		`IN`	`MX`	`0`	`Hamlet.Century.Com`
`*.dev.ibm.com`		`IN`	`MX`	`0`	`Lear.Century.Com`

Fields

name Specifies the full name of the host to which the mail exchanger knows how to deliver mail.

Note: The * (asterisk) in the second *name* entry is a wildcard name entry. It indicates that any mail to the domain `dev.ibm.com` should be routed through the mail gateway `Lear.Century.Com`.

ttl Time to live.

addr-class Internet (IN).

Record Type Mail Exchanger (MX).

Pref value Indicates the order the mailer should follow when there is more than one way to deliver mail to a host.

mailer exchanger
The full name of the mail gateway

Example

The following is an example of a mailing list:

```
dept            IN      MINFO   dept-request jane.merlin.ibm.com
                IN      MG      greg.arthur.ibm.com
                IN      MG      tom.lancelot.ibm.com
                IN      MG      gary.guinevere.ibm.com
                IN      MG      kent.gawain.ibm.com
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in AIX Base Operating System (BOS) Runtime.

Related Information

The **named** daemon.

The DOMAIN Data file, DOMAIN Reverse Data file, DOMAIN Cache file, DOMAIN Local file.

Configuring Name Servers for TCP/IP in the Transmission Control Protocol/Internet Protocol chapter in *Communication Concepts and Procedures*.

tip phones File Format

Purpose

Describes connections used by the **tip** command to contact remote systems.

Description

The **/etc/phones** file lists the remote systems that can be contacted using the **tip** command, and the telephone numbers used to contact those systems.

A sample **phones** file for **tip** is included with AIX. The sample file is named **/etc/phones–file**. A user with root user authority can copy the sample file to the **/etc/phones** file and modify it to suit the needs of a particular site.

Any **tip** user can create an individual phones file in the format of the **phones** file. The individual phones file can be named with any AIX file name and placed in any directory to which the user has access. To instruct the **tip** command to use the new file, set the **tip** command **phones** variable, or set an environment variable named **PHONES**.

Systems listed in the **phones** file must also be described in the **/etc/remote** file or in the file specified by the **tip** command **remote** variable.

Format of Entries

The format of an entry in the **phones** file is as follows:

SystemName PhoneNumber

The *SystemName* field and the *PhoneNumber* field must be separated by at least one space. More than one space can be used to improve readability.

SystemName Specifies the name of the remote system to be contacted.

PhoneNumber Specifies the telephone number, including line access codes, to be used to reach the remote system. Dashes may be used for readability.

If more than one phone number can be used to reach a certain system, make multiple entries for that system, placing each entry on a separate line.

Any line that begins with a # (pound sign) is interpreted as a comment.

Examples

1. To list phone numbers in a **phones** file, make entries similar to the following:

   ```
   hera    1237654
   zeus    9–512–345–9999
   ```

   System `hera` is contacted using the telephone number `123–7654`. To contact system `zeus`, a line access code of `9` is followed by the telephone number `512–345–9999`.

2. To define more than one phone number for the same system, make multiple entries for that system, as follows:

   ```
   decvax    9–915–987–1111
   decvax    9–915–987–2222
   ```

   If the **tip** command cannot reach the `decvax` system using the first phone number, it attempts to contact the system using the second phone number.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/etc/phones	Denotes complete path name of the **phones** file.
/etc/phones–file	Contains an example **phones** file.
/etc/remote	Describes remote systems that can be contacted using the **tip** command.

Related Information

The **tip** command reads the **phones** file.

The tip Overview for System Management in *Communication Concepts and Procedures* discusses configuring the **tip** command.

The **tip** command **phones** variable can be used to specify the phones file to use with the **tip** command.

tip remote File Format

Purpose

Describes remote systems contacted by the **tip** command.

Description

The **/etc/remote** file describes the remote systems that can be contacted using the **tip** command. When a user invokes the **tip** command with the *SystemName* parameter, the **tip** command reads the **remote** file to find out how to contact the specified remote system.

Any **tip** user can create an individual remote file in the format of the **remote** file. The individual remote file can be named with any AIX file name and placed in any directory to which the user has access. To instruct the **tip** command to use the new file, set the **tip** command **remote** variable, or set an environment variable called **REMOTE**.

A sample **remote** file for **tip** is included with AIX. The sample file is named **/etc/remote–file**. This sample file contains *two* examples. One of them uses a set of general dialer definitions, followed by general system definitions, followed by specific systems. The second example defines each system individually.

Any user can copy the sample file to some other directory and modify it for individual use. A user with root user authority can copy the sample file to the **/etc/remote** file and modify it to suit the needs of a particular site.

Format of Entries

The general format of an entry in the **/etc/remote** file is a system name or dialer name followed by a description and one or more attributes, as follows:

SystemName[|*SystemName* ...]|*Description*:*Attribute*[:*Attribute* ...]:

OR

DialerName[|*DialerName* ...]|*Description*:*Attribute*[:*Attribute* ...]:

The name of the system or dialer is followed by a | (pipe symbol) and a description of the system or dialer. More than one name can be given. If multiple names are given, they must be separated by pipe symbols.

The *Description* field is followed by a : (colon) and a list of attributes separated by colons. Each entry must also end with a colon.

An entry can be continued to the next line by typing a \ (backslash). The continuation line must begin with a : (colon), and can be indented for readability. Spaces can be used within the *Description* field *only*.

Any line which begins with a # (pound sign) is read as a comment line.

Attributes Used to Define Systems and Dialers

Use the following attributes to describe systems in the **remote** file:

at=*ACUType* Defines the type of Automatic Call Unit (also known as the ACU or modem). This attribute should be specified in each entry (or in another entry included with the **tc** attribute), unless the system will not be contacted with a modem. The *ACUType* must be one of the following:

biz31f
biz31w
bix22f
biz22w
df02
df03
dn11
ventel
hayes
courier
vadic
v3451
v831

br#*BaudRate* Specifies the baud rate to be used on the connection. The default rate is 300 baud. This attribute should be specified in each entry, or in another entry included with the **tc** attribute. The baud rate specified can be overridden using the **tip** command *–BaudRate* flag.

cu=*Device* Specifies the device for the call unit if it is different from the device defined in the **dv** statement. The default is the device defined in the **dv** statement.

du Flag to make a call. This attribute must be specified in each entry or in another entry included with the **tc** attribute.

dv=*Device*[,*Device* ...]
Lists one or more devices to be used to connect to the remote system. If the first device listed is not available, the **tip** command attempts to use the next device in the list, continuing until it finds one which is available or until it has tried all listed devices.

This attribute must be specified in each entry or in another entry included with the **tc** attribute.

el=*Mark* Defines the mark used to designate an end-of-line in a file transfer. This setting is the same as that defined by the **tip** command **eol** variable.

fs=*Size* Specifies the frame size. The default is the value of the **BUFSIZ** environment variable. This value can also be changed using the **tip** command **framesize** variable.

ie=*InputString* Specifies the input end-of-file mark. The default is NULL.

oe=*OutputString*
Specifies the output end-of-file mark. The default is NULL.

pn= Lists telephone numbers to be used to contact the remote system. This entry is required if a modem is to be used to contact the remote system.

pn=@ Instructs **tip** to search the **/etc/phones** file, or the file specified with the **phones** variable, for the telephone number.

pn=*Number*[,*Number* ...] Lists one or more phone numbers to be used to contact the remote system.

tc=*Entry* Refers to another entry in the file. This allows you to avoid defining the same attributes in more than one entry. If used, this attribute should be at the end of the entry.

tc=*DialerName* Includes the specified *DialerName* entry in this entry. The *DialerName* must be defined elsewhere in the **remote** file.

tc=*SystemName* Includes the specified *SystemName* entry in this entry. The *SystemName* must be defined elsewhere in the **remote** file.

Ways to Group Entries

There are two ways to set up entries in the **remote** file.

One is to simply define each system individually, giving all of its attributes in that entry. This works well if you are contacting several dissimilar systems.

Another way to set up the file is to group systems by similarity. To do this, use two or three sections, depending on the ways the systems are similar. You can use:

- Dialer definitions, including the device, baud rate, call unit, ACU type, and dial up flag
- General system definitions, including any information that several systems have in common and using the **tc** attribute to refer to a dialer entry
- Specific system descriptions, which use the **tc** attribute to refer back to one of the general system types or to a dialer entry.

You can omit either the dialer definitions or the general system definitions, depending on the way you group the remote systems.

Examples

Defining a System Individually

1. To define a system without using the **tc=** attribute, make an entry similar to the following:

```
vms750|nymph|NPG 750:\
        :dv=/dev/tty36,/dev/tty37:br#9600:el=^Z^U^C^S^Q^O:\
        :ie=$@:eo=^Z:
```

This entry defines system `vms750`, which can also be referred to as `nymph`. The system can be accessed using either `/dev/tty36` or `/dev/tty37`, at a baud rate of `9600`. The end-of-line mark is `^Z^U^C^S^Q^O`. The input end-of-file mark is `$@` and the output end-of-file mark is `^Z`. Since no phone number is defined, the system is accessed over a direct connection.

Grouping Systems by Similarity

The following examples use a dialer entry, a general system entry, and specific system entries which refer to the general entries.

2. To define a dialer, make an entry similar to the following:

```
dial1200|1200 Baud Able Quadracall attributes:\
        :dv=/dev/cul1:br#1200:at=dn11:du:
```

   This entry defines a dialer called `dial1200`. The dialer is connected to device `/dev/cul1`, is an ACU type of `dn11`. The dial-up (`du`) flag is set.

3. To define a general system type and refer to a dialer entry, make an entry similar to the following:

```
unix1200|1200 Baud dial-out to another UNIX system:\
        :el=^U^C^R^O^D^S^Q:ie=%$:oe=^D:tc=dial1200:
```

   This entry defines a system type called `unix1200`. The end-of-line mark for communication with this type of remote system is `^U^C^R^O^D^S^Q`. The input end-of-file mark is `%$` and the output end-of-file mark is `^D`. The dialer defined by the `dial1200` entry is used.

4. To describe a specific system, make an entry similar to the following:

```
zeus|CSRG ARPA VAX-11/780:pn=@:tc=unix1200:
```

   This entry describes system `zeus`. The **tip** command will search the **/etc/phones** file for the telephone number and use the attributes for a `unix1200` system type.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Files

/etc/remote	Denotes complete path name of the **remote** file.
/etc/phones	Lists phone numbers used to contact remote systems.
/etc/remote-file	Contains an example **remote** file.

Related Information

The **tip** command reads the **remote** file.

The tip Overview for System Management in *Communication Concepts and Procedures* discusses setting up a **remote** file.

The **tip** command **remote** variable, can be used to specify the remote file to use with the **tip** command.

tip .tiprc File Format

Purpose

Provides initial settings of variables for the **tip** command.

Description

The **.tiprc** file allows you to initialize variable settings for the **tip** command. When first invoked, the **tip** command checks the user's home directory (defined by the **$HOME** environment variable) for a **.tiprc** file. If the file is present, the **tip** command sets the **tip** variables according to instructions in the **.tiprc** file.

The **tip** command uses several different types of variables: numeric, string, character, or Boolean. A Boolean variable can be toggled by putting the variable name in the **.tiprc** file, or it can be reset by putting an ! (exclamation point) in front of the variable name. Other types of variables are set by following the variable name with an = (equal sign) and the new value of the variable.

You can use the **–v** flag of the **tip** command to see the variable settings as they are made. Also, you can use the **~s** escape signal to change variables while the **tip** command is running.

Example

Following is a sample **.tiprc** file:

```
be
ba=9600
!echocheck
```

This file toggles the **beautify** variable, sets the **baudrate** variable to 9600, and resets the **echocheck** variable to the default setting.

Implementation Specifics

This file is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

File

$HOME/.tiprc	Complete path name of the **.tiprc** file.

Related Information

The **tip** command uses the **.tiprc** file.

The tip Overview for System Management in *Communication Concepts and Procedures* discusses setting up a **.tiprc** file.

tip Command Variables describes the variables that can be set in the **.tiprc** file. Settings made with tip Command Escape Signals can override settings made in the **.tiprc** file.

Chapter 3. Special Files

Special Files Overview

A *special file* is file that is associated with a particular hardware device or other resource of the computer system. AIX uses special files, sometimes called device files, to provide file I/O access to specific character and block device drivers. Special files, at first glance, appear to be just like ordinary files:

- They have path names that appear in a directory.
- They have the same access protection as ordinary files.
- They can be used in almost every way that ordinary files can be used.

However, there is an important difference between the two. An ordinary file is a logical grouping of data recorded on disk. A special file, on the other hand, corresponds to a device entity. Examples are:

- An actual device, such as a line printer
- A logical subdevice, such as a large section of disk drive
- A pseudo device, such as the physical memory of the computer (**/dev/mem**) or the null file (**/dev/null**).

The special files are distinguished from other files by having a file type (c or b, for character or block) stored in their i-nodes to indicate the type of device access provided. The i-node for the special file also contains the device major and minor numbers assigned to the device at device configuration time.

Warning: POTENTIAL FOR DATA CORRUPTION OR SYSTEM CRASHES: Data corruption, loss of data, or loss of system integrity will occur if devices supporting paging, logical volumes, or mounted file systems are accessed using block special files. Block special files are provided for logical volumes and disk devices on AIX and are solely for system use in managing filesystems, paging devices and logical volumes. They should not generally be used for other purposes. Additional information concerning the use of special files may be obtained in Understanding I/O Access through Special Files.

Several special files are provided with the AIX operating system. By convention, the special files are located in the **/dev** directory. More information about the following special files is provided in this documentation:

3270cn	Provides access to IBM 3270 Connection Adapters by way of the IBM 3270 Connection Adapter device driver.
bus	Provides access to the hardware bus by way of the machine I/O device driver.
cd	Provides access to the **cdrom** device driver.
console	Provides access to the system console.
dump	Supports system dump.
entn	Provides access to the 3COM Ethernet adapters by way of the RISC System/6000 Ethernet device handler.
error	Supports error logging.
fd	Provides access to the diskette device driver.
hft	Implements a high function terminal (HFT) device.

hia0 Provides access to the IBM Host Interface Adapter (HIA) by way of the IBM HIA device handler.

kmem and **mem**
Provides privileged virtual memory read and write access.

lp Provides access to the line printer device driver.

lvdd Provides access to the logical volume device driver.

mpqn Provides access to multiprotocol adapters by way of the Multiprotocol (MPQP) device handler.

null Provides access to the null device.

nvram Provides access to platform-specific nonvolatile RAM used for system boot, configuration, and fatal error information.

pty Provides the pseudo terminal (PTY) device driver.

rhdisk Provides raw access to the physical volume (fixed-disk) device driver.

rmt Provides access to the sequential access bulk storage medium device driver.

SCSI Provides access to the SCSI adapter driver.

tokn Provides access to the Token-Ring adapters by way of the Token-Ring device handler.

trace Supports event tracing.

tty Supports the controlling terminal interface.

Related Information

The Header Files Overview, which describes header files in general and lists header files discussed in this documentation.

The File Formats Overview, which defines and describes file formats in general and lists file formats discussed in this documentation.

The discussion of Device Driver Classes, Device Driver Roles, and File I/O Access Through Special Files in *General Concepts and Procedures*.

3270cn Special File

Purpose

Provides access to IBM 3270 Connection Adapters by way of the IBM 3270 Connection Adapter device handler.

Description

The **3270c***n* character special file provides access to the IBM 3270 Connection Adapter device handler for the purpose of emulating 3270 display stations and printers. The device handler is a multiplexed device handler that supports an independent, logical 3270 session on each of its channels.

The device handler supports two modes of operation:

- Non-SNA Distributed Function Terminal (DFT) mode

  In DFT mode, the adapter can appear as multiple terminal sessions, or printer sessions, or both, and is an intelligent device to the control unit. In this mode, the device handler provides the capability of emulating several IBM 3278/79 display stations. If the attached control unit does not support Extended Asynchronous Event Device Status, the control unit port must be configured for one session only *or* the device handler must be configured for one session only.

- 3278/79 emulation Control Unit Terminal (CUT) mode

  In CUT mode, the adapter appears as a single-session, unintelligent device to the control unit. In this mode, the device handler provides the capability of emulating a single IBM 3278/79 display station.

The device handler supports up to four IBM 3270 Connection adapters, each of which may have up to 5 DFT sessions or 1 CUT session.

The **<sys/io3270.h>** file contains the definitions of the structures used by the device handler.

Usage Considerations

When accessing the IBM 3270 Connection device handler, the following should be taken into account:

- Driver Initialization and Termination

  The device handler may be loaded and unloaded. The device handler supports the configuration calls to initialize and terminate itself, but does not support the configuration call to query vital product data (VPD).

- Special File Support

  Subroutines other than the **open** call and the **close** call are discussed in regard to the mode in which the device handler is operating.

Subroutine Support

The 3270 device handler provides 3270-specific support for the following subroutines:

- **open**
- **close**
- **read**
- **readx**
- **write**
- **writex**
- **ioctl**

The open and close Subroutines

The device handler supports the **/dev/3270cn** special file as a character-multiplex special file. The special file must be opened for both reading and writing (O_RDWR).

A special consideration exists for closing the **3270cn** special file. If the file was opened in both CUT mode and CUT File Transfer mode, the **close** operation for CUT File Transfer mode must be done before the **close** operation for CUT mode

The special file name used in an **open** call takes on several different forms depending on how the device is to be opened. Types of special file names are:

dev/3270c*n*/C Starts the device handler in CUT mode for the selected port, where the value of n is 0 <= *n* <= 7.

/dev/3270c*n*/F Starts the device handler in CUT File Transfer mode for the selected port, where the value of *n* is 0 <= *n* <= 7. The file must be currently open in CUT mode before it can be opened in CUT File Transfer mode.

/dev/3270c*n*/* Starts the device handler in DFT mode for the selected port, where the value of n is 0 <= n <= 7 and * is defined as one of the following:

P/00, P/01, P/02...P/1F
: Printer session with the *P* variable is equal to the control unit session address, and the value of *a* is less than or equal to 0x1F.

01 to 05
: Terminal session number.

/dev/3270c*n* Starts the device handler in DFT mode for the selected port, where the value of n is 0 <= n <= 7.

The read Subroutine in DFT Mode

Data received by the communication adapter from the host is placed in the buffer until the message completes or the buffer is full. When either condition occurs, the AIX driver returns program control back to the application. The application can determine the status of a **read** subroutine call by issuing a WDC_INQ **ioctl** subroutine.

If the WDC_INQ **ioctl** subroutine returns a status that indicates there is more data available, the application should immediately issue another **read** call. Available data must be read as soon as possible so as not to degrade link or host performance.

If a **read** subroutine call is made and no data is available, the calling process is blocked until data becomes available. To avoid blocking, use the **poll** subroutine to determine if data is available.

The host sends data as an outbound 3270 data stream. The device handler translates the command codes in the outbound 3270 data stream. The command codes and translations are as follows:

Command Code Translation Table		
Command Code	**Into Driver**	**Out of Driver**
Erase All Unprotected	0x6F	0x0F
Erase/Write	0xF5	0x03
Erase/Write Alternate	0x7E	0x0D
Read Buffer	0xF2	0x02
Read Modified	0xF6	0x06
Write	0xF1	0x01
Write Structured Field	0xF3	0x11

Note: The 3270 write commands require the application to send status to the host indicating if the 3270 data stream is valid. Status may be sent using the WDC_SSTAT **ioctl** subroutine.

The readx Subroutine in DFT mode

Data received by the communication adapter from the host is placed in the buffer until either the message completes or the buffer is full. Upon completion of the **read** operation, the **io3270** structure pointed to by the **read** extension contains the status. One of the following status codes is set in the **io_flags** field of the **io3270** structure:

WDI_DAVAIL Indicates that there is additional data for this link address.

WDI_COMM Indicates that there is a communication error; the **io_status** field contains the corresponding message code.

WDI_PROG Indicates that there is a program error; the **io_status** field contains the corresponding message code.

WDI_MACH Indicates that there is a hardware error; the **io_status** field contains the corresponding message code.

WDI_FATAL Indicates that an error occurred which prevents further communication with the host. This flag is set in addition to the **WDI_COMM** flag, **WDI_PROG**, or **WDI_MACH** flags. It is also set is a COAX failure occurs. In this case the **io_status** field contains WEB_603, but the **WDI_COMM**, **WDI_PROG**, or **WDI_MACH** flags are not set.

When reset, the **WDI_DAVAIL** flag indicates that the data just read marks the completion of an outbound 3270 data stream.

If the **WDI_DAVAIL** flag indicates there is more data available, another **readx** subroutine call should be issued immediately. Available data must be read as soon as possible so as not to degrade link or host performance.

If a **readx** subroutine call is made and no data is available, the calling process is blocked until data becomes available. To avoid blocking, use the **poll** subroutine to determine if data is available.

Data received from the host is in the form of an outbound 3270 data stream. The device driver translates the command codes in the outbound 3270 data stream.

Note: The 3270 write commands require the application to send a status to the host. Status may be sent using the WDC_SSTAT **ioctl** subroutine.

The write Subroutine in DFT Mode

The **write** subroutine sends an inbound 3270 data stream to the host. The buffer specified on a **write** subroutine must contain a complete inbound 3270 data stream. The **write** operation completes when the data has been successfully transferred from the buffer specified on the subroutine call.

The writex Subroutine in DFT mode

The **writex** subroutine sends to the host an inbound 3270 data stream. The buffer specified on a **writex** subroutine call must contain a complete inbound 3270 data stream.

The **write** subroutine completes when it successfully transfers the data from the specified buffer. Upon completion of the **write** subroutine call, the **io3270** structure pointed to by the **write** extension contains the status. One of the following status code is set in the in the **io_flags** field of the **io3270** structure:

WDI_DAVAIL Indicates that there is data available for this link address; the data must be read before any write can occur.

WDI_COMM Indicates that there is a communication error; the **io_status** field contains the corresponding message code.

WDI_PROG Indicates that there is a program error; the **io_status** field contains the corresponding message code.

WDI_MACH Indicates that there is a hardware error; the **io_status** field contains the corresponding message code.

The ioctl Subroutine in DFT Mode

The **ioctl** subroutine may be issued to the device handler when it is in DFT mode. The following are the available **ioctl** commands:

IOCINFO Returns the logical terminal number. This number is the EBCDIC representation of the controller type, and the controller attachment protocol in the **iocinfo** structure.

WDC_AUTO Provides the handler with the option to automatically acknowledge the receipt of a valid 3270 data steam. An acknowledgment is sent only if the beginning of the 3270 data stream consists of 0xF3 00 06 40 00 F1 C2 10 14. This command also allows the driver to indicate no acknowledgment upon receipt of data.

WDC_INQ Queries the status of the last **read** or **write** call issued by the application. Also, WDC_INQ is used to determine if data is available for reading. The status is placed in the **io_flags** field of the **io3270** structure. This field accepts the following values:

WDI_DAVAIL Data is available for reading. The data are either buffered in the driver or in the communication adapter. The data should be read immediately so as to not impact performance. A **write** or **writex** subroutine call cannot be completed until the data is read.

WDI_COMM, WDI_PROG, or **WDI_MACH**
Indicate a communication check, program check, or machine check, respectively. In that case, the **io_status** field contains a message code specifying the type of check.

WDC_POR The link address is disabled and then re–enabled to emulate a 3270 terminal power–on reset function

WDC_SSTAT Sends status to the host. The argument field contains one of the following values:

- **STAT_ACK** The previously received 3270 data stream is valid and the proper response is made to the host.
- **STAT_RESET** Sends a RESET KEY to the DFT device handler.
- **STAT_PRTCMP** The printer session has completed printing the data.
- **STAT_BERR** A bad buffer order or an invalid buffer address was received.
- **STAT_BADC** A non–3270 command was received.
- **STAT_UNSUP** An unsupported 3270 command was received.

The **<sys/io3270.h>** header file contains the definitions of the structures used by the device handler.

Error Conditions in DFT Mode

The following error conditions may be returned when accessing the device handler through the **/dev/3270cn** special file:

EBUSY Indicates that an open was requested for a channel that is already open.

EFAULT Indicates that a buffer specified by the caller was invalid.

EINTR Indicates that a subroutine call was interrupted.

EINVAL Indicates that an invalid argument was received.

EIO Indicates that an unrecoverable I/O error occurred on the requested data transfer.

ENODEV Indicates that an open was requested for an invalid channel.

ENOMEM Indicates that the driver could not allocate memory for use in the data transfer.

ENXIO Indicates that an operation was requested for an invalid minor device number.

The read Subroutine in CUT Mode

The **read** subroutine places data received by the communication adapter in a buffer.

Note: To set the offset into the communication adapter's buffer from which to read, use the EMSEEK ioctl operation.

Two **ioctl** commands control the way the **read** subroutine operates the EMNWAIT and EMWAIT operations. The EMNWAIT operation indicates that subsequent read calls should be satisfied immediately. The EMWAIT **ioctl** operation (the default) indicates that read calls should be satisfied only after an interrupt from the control unit indicates that something has changed on the display. The following are control unit interrupts:

Buffer Modification Complete
: The **read** subroutine returns the number of bytes requested.

Load I/O Address Command Decoded
: The **read** subroutine returns 0 (zero) for the number of bytes read.

Write Subroutine in CUT Mode

The **write** subroutine sends an inbound 3270 data stream to the host. The buffer specified on a **write** subroutine must contain a complete inbound 3270 data stream. To set the offset into the communication adapter's buffer to begin to write, use the EMSEEK **ioctl** operation.

The ioctl Subroutine in CUT Mode

The **ioctl** subroutine may be issued to the device handler in CUT mode. The following are acceptable ioctl operations:

EMKEY Sends a scancode to the emulation adapter. The scancode is logically ORed with the EMKEY operation and the result is used as the command field on the **ioctl** subroutine call.

EMCPOS Returns the position of the cursor relative to the start of the communication adapter's buffer.

EMXPOR Disables the link address and then re–enables it to emulate a 3270 terminal power–on reset function.

EMNWAIT Specifies that **read** subroutine calls should be satisfied immediately.

EMWAIT Specifies that **read** subroutine calls should be satisfied only after a change to the emulation buffer or the cursor position (this is the default setting).

EMVISND Returns the current contents of the emulation Visual/Sound register in the **integer** field. The address of this field is specified as the argument to the EMVISND operation.

EMIMASK Provides a mask to specify which interrupts appear. The argument field specifies the address of the mask. The low–order bits of the mask (0–7) correspond to bits 0 – 7 of the Interrupt Status register. Bits 8 – 15 of the mask correspond to bits 0 – 7 of the Visual/Sound register.

This operation allows the driver to ignore visual or sound interrupts except for those bits specifically masked ON. When a bit is on, the interrupt that corresponds to that bit position appears. Interrupts that correspond to off (0) bit positions in the mask are discarded by the device handler. The previous mask setting is returned to the caller in the mask field. The interrupt status bits and the visual or sound bits are documented in the *IBM 3270 Connection Technical Reference*.

IOCINFO Returns a structure of device information, as defined in the **devinfo.h** file, to the user–specified area. The *devtype* field is DD_EM78, as defined in the **<sys/devinfo.h>** file, and the flags field is 0 (zero).

EMSEEK Sets the offset into the communication adapter's buffer to begin a **read** operation or **write** operation.

Error Conditions in CUT Mode

The following error conditions may be returned when accessing the device handler through the **dev/3270cn** special file:

EBUSY An open was requested for a channel that is already open. The keystroke buffer is full.

EFAULT A buffer specified by the caller is invalid.

EINTR A subroutine call was interrupted.

EINVAL An invalid argument was specified on an **ioctl** request.

EL3RST A reset command was received by the communications adapter.

ENOCONNECT The connection to the control unit went down while a **read** operation for which the EMWAIT **ioctl** operation had been specified was waiting.

EIO An unrecoverable I/O error occurred on the requested data transfer.

ENXIO An operation was requested for an invalid minor device number.

Implementation Specifics

This special file requires the IBM 3270 Connection Adapter.

Files

/usr/include/sys/io3270.h Defines structures used by the device handler.

usr/include/sys/devinfo.h Defines device information structures.

/dev/3270c0/*.../dev/3270c7/*

Related Information

The **close** subroutine, **poll** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files in *Kernel Extensions and Device Support Programming Concepts*.

Understanding Major and Minor Numbers For A Special File in *Kernel Extensions and Device Support Programming Concepts*.

bus Special File

Purpose

Provides access to the hardware bus by way of the machine I/O device driver.

Description

The **bus** special file consists of a pseudo-driver in the kernel that allows a privileged user to access the hardware I/O bus. This is done indirectly by use of the **ioctl** subroutine. The calling process must have the appropriate system privilege to open the **bus** special file.

This capability should be used only by device initialization and configuration programs. Programs that depend upon the **bus** device interface may not be portable to machines with different hardware.

Implementation Specifics

For additional information concerning the use of the **/dev/bus** special file, refer to the Machine Device Driver documentation in the Device Configuration Subsystem: Programming Introduction.

File

/dev/bus0

Related Information

The **ioctl** subroutine.

The Device Configuration Subsystem: Programming Introduction in *Kernel Extensions and Device Support Programming Concepts*.

The Special Files Overview, which presents general information about spec ial files.

Understanding I/O Access Through Special Files in *Kernel Extensions and Device Support Programming Concepts*.

Understanding Device Driver Classes in *Kernel Extensions and Device Support Programming Concepts*.

Understanding Character I/O Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

Understanding Pseudo-Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

cd Special File

Purpose

Provides access to the **cdrom** device driver.

Description

The **cdrom** special file provides block and character (raw) access to disks in the **cdrom** drives. Compact disks are read-only media that provide storage for large amounts of data. Block access to compact disks is achieved through the special files **/dev/cd0**, ... Character access is provided through the special files **/dev/rcd0**, ...

The prefix of r on a special file name means that the drive is accessed as a raw device rather than a block device. Performing raw I/O with a compact disk requires that all data transfers be in multiples of the compact disk logical block length. Also, all **lseek** subroutines that are made to the raw **cdrom** device driver must set the file offset pointer to a value that is a multiple of the specified logical block size.

Device-Dependent Subroutines

Most **cdrom** operations are implemented using the **open**, **read**, and **close** subroutines. However, for some purposes, use of the **openx** (extended) subroutine is required.

The open and close Subroutines

The **openx** subroutine is supported to provide additional functions to the **open** sequence. The **openx** operation requires appropriate authority to execute. Attempting to execute this subroutine without the proper authority results in a return value of –1 with the **errno** global variable set to the EPERM value.

The ioctl Subroutine

One **ioctl** operation is defined for all device drivers that use the **ioctl** subroutine. This is the IOCINFO operation. The remaining **ioctl** operations are all physical volume device-specific operations. Diagnostic mode is not required for the following operation.

The IOCINFO ioctl operation

The IOCINFO **ioctl** operation returns a **devinfo** structure,which is defined in the **<sys/devinfo.h>** header file.

Error Conditions

In addition to those errors listed for the **ioctl**, **open**, **read**, and **write** subroutines, the following other error types are possible:

EACCES	A subroutine other than **ioctl** or **close** was attempted while in diagnostic mode.
EACCES	A normal **read** call was attempted while in diagnostic mode.
EFAULT	Illegal user address.
EBUSY	The target device is reserved by another initiator.
EINVAL	The device has been opened with a mode other than read-only.
EINVAL	An *nbyte* parameter to a **read** subroutine is not an even multiple of the blocksize.

EINVAL	A sense-data buffer length greater than 255 is invalid for a CDIOCMD **ioctl** operation.
EINVAL	A data buffer length greater than **SC_MAXREQUEST** is invalid for a CDIOCMD **ioctl** operation.
EINVAL	An attempt has been made to configure a device that is still open.
EINVAL	An illegal configuration command has been given.
EMFILE	An **open** operation has been attempted for a SCSI adapter which already has the maximum permissible number of open devices.
ENOTREADY	There is no compact disk in the drive.
ENODEV	An attempt has been made to access a device that is not defined.
ENODEV	An attempt has been made to close a device that has not been defined.
EMEDIA	The media has been changed.
EIO	Hardware error or aborted command or illegal request.
EIO	An attempt has been made to read beyond the end of media.
EPERM	This subroutine requires appropriate authority.
ESTALE	A **cdrom** disk has been ejected (without first being closed by the user) and then either re-inserted or replaced with a second disk.
ETIMEDOUT	An I/O operation has exceeded the given timer value.

Implementation Specifics

The SCSI **cdrom** Device Driver provides more information about implementation specifics.

The **cd** special file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/cd0, ...

/dev/rcd0, ...

Related Information

The SCSI **cdrom** device driver.

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **lseek** subroutine, **ioctl** subroutine.

SCSI Subsystem Programming Introduction in *Kernel Extensions and Device Support Programming Concepts*.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files in *Kernel Extensions and Device Support Programming Concepts*.

Understanding Block I/O Device Drivers, Providing Raw I/O Access in a Block Device Driver in *Kernel Extensions and Device Support Programming Concepts*.

console Special File

Purpose

Provides access to the system console.

Description

The /**dev/console** special file provides access to the device or file designated as the system console. This file can be designated as the console device by the person administering the system or a user with appropriate privilege. The **console** character special file provides access to the console device driver. The console device driver in turn directs input and output to the device or file selected as the system console.

The system console provides two functions in the AIX operating system. It is typically a terminal or display located near the system unit and provides access to the system when it is operating in a non-multi-user mode. (This would be the case for maintenance and diagnostic purposes.) A console login is also normally provided on this device for all operating system run levels.

The second function of the system console is to display messages for system errors and other problems requiring intervention. These messages are generated by the operating system and its various subsystems when booting or operating. The system console can also be redirected to a file or to the /**dev/null** special file for systems operating without a console device.

Console Configuration Support

The console driver configuration support allows the system console to be assigned or reassigned to a specified device or file. The console configuration support also provides query functions to determine the current and configured path names for the device or file designated as the console. This configuration support is used by the **swcons**, **chcons**, and **lscons** high-level system management commands. It is also used by the console configuration method at system boot time.

The **swcons** (switch console) command can be used during system operation to switch the system console output to a different target temporarily. This command switches only system information, error, and intervention-required messages to the specified destination. The **swcons** command does not affect the operation of the system console device providing a login through the **getty** command. The device or file specified when using the **swcons** command remains the target for console output until one of the following happens:

- Another **swcons** command is issued.
- The system is rebooted.
- The console driver detects an error when accessing the designated device or file.

If an open or write error is detected on the device or file specified by the **swcons** command, the console device driver switches all output back to the device or file providing console support when the system booted.

The **chcons** (change console) command can be used to switch the system console output to a different device or file for the next reboot. This command does not affect the current console selection, but becomes effective when the system is re-booted.

When requested to activate a login on the console device, the **getty** program (which provides login support) uses the console configuration support to determine the path name of the targeted console device used at boot time. This ensures that the **swcons** command does not effect the console device being used for login.

Usage Considerations

The **open**, **close**, **read**, **write**, **ioctl**, **select** and **poll** subroutines are supported by the console device driver and may be used with the **/dev/console** special file. These subroutines are redirected to the device or file serving as the current system console device by the console device driver.

The open and close Subroutines

When an **open** subroutine call is issued to the console device driver, it is redirected to the device or file currently chosen as the console device. If the system console choice is a file, the file is opened with the *append* and *create* options when the first open of the **dev/console** file is received. Subsequent opens have no effect when the console selection is a file. However, they are passed on to the device driver supporting the device chosen as the console.

If the console selection has been temporarily switched using the **swcons** command and the first open of the new underlying device fails, the console device driver switches back to the console device or file with which the system was booted. This prevents important system messages from being lost.

If an *ext* parameter is passed using the **openx** subroutine, it is either passed to the device driver supporting the console target, or is ignored. (The latter is true if the console selection is a file.)

The **close** subroutine support is standard.

The select, poll, and ioctl Subroutines

The **select**, **poll** and **ioctl** subroutines are redirected to the current system console device when the console selection is not a file. If the selected console device is a file, the console device driver returns an error indicating that the subroutine is not supported.

If an *ext* parameter is passed to the **ioctlx** subroutine, it is then passed to the device driver supporting the console target, or else ignored. (The latter is true if the console selection is a file.)

The read and write Subroutines

Write subroutine calls are redirected to the current console device or file. If the console selection has been temporarily switched using the **swcons** command, and the write to the targeted device or file fails, the console device driver switches back to the console device or file with which the system was booted and retries the write. This prevents important system messages from being lost in case the temporary console target is unavailable or fails. The console device driver stays switched to the device the system was booted with until another **swcons** command is issued.

The **read** subroutine calls are redirected to the current console selection if it is a device. If the current console selection is a file, the **read** is rejected with an error (EACCES).

If an *ext* parameter is passed to the **readx** or **writex** subroutine, it is passed to the device driver supporting the console target, or is ignored. (The latter is true if the console selection is a file.)

Implementation Specifics

The **console** special file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/console

/dev/null

Related Information

The **swcons** command, **chcons** command, **lscons** command, **getty** command.

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **lseek** subroutine, **ioctl** subroutine, **select** subroutine, **poll** subroutine.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files, Understanding Character I/O Device Drivers and Understanding Pseudo-Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

dump Special Files

Purpose

Supports system dump.

Description

The **/dev/sysdump** and **/dev/sysdumpctl** special files support system dumping. Minor device 0 of the **sysdump** driver provides the interfaces for the system dump routine to write data to the dump device. The **sysdump** driver also provides interfaces for querying or assigning the dump devices and initiating a dump.

Implementation Specifics

The **dump** special files are part of AIX Base Operating System (BOS) Runtime.

Files

/dev/sysdump	The path to the **sysdump** special file.
/dev/sysdumpf	The path to the **sysdumpf** special file.
/usr/include/sys/dump.h	The path to the **dump.h** header file.

Related Information

The **dmp_add** kernel service, **dmp_del** kernel service.

The Special Files Overview, which presents general information about special files.

RAS Kernel Services, Understanding I/O Access Through Special Files, Understanding Major and Minor Numbers, Understanding Device Driver Classes, and Understanding Pseudo-Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

entn Special File

Purpose

Provides access to Ethernet High-Performance LAN adapters by way of the RISC System/6000 Ethernet device handler.

Description

The **/dev/ent***n* character special file provides access to the Ethernet device handler for the purpose of providing access to an ethernet Local Area Network. The device handler supports up to four adapters, each of which can be running either or both the standard ethernet or IEEE 802.3 protocols.

Usage Considerations

When accessing the Ethernet device handler, the following should be taken into account:

- Driver Initialization and Termination

  The device handler may be loaded and unloaded. The handler supports the configuration calls to initialize and terminate itself.

- Special File Support

  Calls other than the **open** and **close** subroutine calls are discussed based on the mode in which the device handler is operating.

Subroutine Support

The Ethernet device handler supports the **open**, **close**, **read**, **write**, and **ioctl** subroutines in the following manner:

- The **open** and **close** subroutines

  The device handler supports the **/dev/ent***n* special file as a character–multiplex special file. The special file must be opened for both reading and writing (O_RDWR). There are no particular considerations for closing the special file. The special file name used in an **open** call differs depending upon how the device is to be opened. Types of special file names are:

  /dev/ent*n* An **open** subroutine call to this device is used to start the device handler for the selected port, where the value of *n* is 0 <= *n* <= 7.

  /dev/ent*n***/D** An **open** subroutine call to this device is used to start the device handler for the selected port in diagnostic mode, where the value of *n* is 0 <= *n* <= 7.

- The **read** subroutine

  Can take the form of a **read, readx**, **readv**, or **readvx** subroutine. For this call, the device handler copies the data into the buffer specified by the caller.

- The **write** subroutine

  Can take the form of a **write**, **writex**, **writev**, or **writevx** subroutine. For this call, the device handler copies the user data into a buffer and transmits the data on the LAN.

- The **ioctl** subroutine

  The Ethernet device handler supports the following **ioctl** operations:

  CIO_START Starts a session and registers a network ID.

  IOCINFO Returns a device information structure to the user specified area. The **devtype** field is DD_NET_DH and the **devsubtype** field is DD_EN, as defined in the **<sys/devinfo.h>** file.

  CIO_HALT Halts a session and unregisters a network ID.

  CIO_QUERY Returns the current RAS counter values, as defined in the **<sys/comio.h>** and **<sys/entuser.h>** files.

  CIO_GET_STAT Returns current adapter and device handler status.

  CIO_GET_VPD Returns adapter vital product data if available and valid.

  ENT_SET_MULTI Sets or clears a multicast address.

Error Conditions

The following error conditions may be returned when accessing the device handler through the **dev/ent***n* special file:

EACCES Indicates that permission to access the port is denied for one of the following reasons:

- The device has not been initialized.
- The request to open the device in Diagnostic mode is denied.
- The call is from a kernel mode process.

EAFNOSUPPORT Indicates that the address family is not supported by the protocol or that the multicast bit in address is not set.

EAGAIN Indicates that the transmit Queue is full.

EBUSY Indicates that the request is denied because the device is already opened in Diagnostic mode or the maximum number of opens was reached.

EEXIST Indicates that the define device structure (DDS) already exists.

EFAULT Indicates that an invalid address or parameter was specified.

EINTR Indicates that a subroutine call was interrupted.

EINVAL Indicates that an invalid range, operation code was specified or that the device is not in Diagnostic mode.

EIO Indicates that an I/O error occurred.

ENOBUFS Indicates that no buffers are available.

ENOCONNECT Indicates that a connection was not established.

ENODEV Indicates that the device does not exist.

ENOENT Indicates that there is no DDS to delete.

ENOMEM Indicates that the device does not have enough memory.

ENOMSG Indicates that no message of desired type was available.

ENOSPC Indicates that there is no space left on device (the Multicast table is full).

ENOTREADY Indicates that the device is not ready, a CIO_START operation was not issued, or was issued and did not complete.

ENXIO Indicates that the device does not exist or that the maximum number of adapters was exceeded.

EUNATCH Indicates that the protocol driver is not attached.

Implementation Specifics

This file functions with the Ethernet device handler.

File

/usr/include/sys/devino.h Contains device types and information.

Related Information

The **open** subroutine, **close** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine.

The Special Files Overview, which presents general information about special files.

Ethernet Device Handler Overview in *Communication Concepts and Procedures.*

Understanding I/O Access to Special Files, Understanding Major and Minor Numbers For A Special File, Understanding Raw I/O Access to Block Special Files in *Kernel Extensions and Device Support Programming Concepts.*

error Special File

Purpose

Supports error logging.

Description

The **error** and **errorctl** special files support logging of error events. Minor device 0 of the **error** special file is the interface between processes that log error events and the error daemon. Error records are written to the **error** special file by the **errlog** library routine and the **errsave** kernel service. The **error** special file timestamps each error record entry.

The error daemon opens **error** file for reading. Each read retrieves an entire error record. The format of error records is described in the **erec.h** header file.

Each time an error is logged, the error ID, the resource name, and the timestamp are recorded in nonvolatile random access memory (NVRAM) so that in the event of a system crash, the last error logged is not lost. When the **error** file is restarted, the last error entry is retrieved from NVRAM.

The standard device driver interfaces (open, close, read, and write) are provided for the **error** file. The **error** file has no **ioctl** functions.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/error The path to the **error** special file.
/dev/errorctl The path to the **errorctl** special file.
/usr/include/sys/err.h
The path to the **err.h** header file.
/usr/include/sys/erec.h
The path to the **erec.h** header file, which describes the format of error records .

Related Information

The **errclear** command, **errdead** command, **errdemon** command, **errinstall** command, **errlogger** command, **errmsg** command, **errpt** command, **errstop** command, **errupdate** command.

The **errsave** kernel service.

The **errlog** subroutine.

The Special Files Overview, which presents general information about special files.

The discussion of RAS Kernel Services in *Kernel Extensions and Device Support Programming Concepts*.

fd Special File

Purpose

Provides access to the diskette device driver.

Description

The **fd** special file provides block and character (raw) access to diskettes in the diskette drives. The special file name can specify both the drive number and the format of the diskette. The exceptions are **/dev/fd0** and **/dev/fd1**, which specify diskette drives 0 and 1, respectively, without specifying their formats.

The generic special files **/dev/rfd0** and **/dev/rfd1** determine the diskette type automatically for both drive 0 and drive 1. First, the device driver attempts to read the diskette using the characteristics of the default diskette for the drive type. If this fails, the device driver changes its characteristics and attempts to read either until it has read the diskette successfully or until it has tried all the possibilities supported for the drive type by the device driver.

A prefix of **r** on a special file name means that the drive is accessed as a raw device rather than a block device. Performing raw I/O with a diskette requires that all data transfers be in multiples of the diskette sector length. Also, all **lseek** subroutine calls made to the raw diskette device driver must result in a file offset value that is a multiple of the sector size. For the diskette types supported, the sector length is always 512 bytes.

Note: The diskette device driver does not perform read verification of data that is written to a diskette.

Types of Diskettes Supported

The 1.2 M-byte, 5.25-inch diskette drive and the 1.44 M-byte, 3.5-inch diskette drive are both supported. All **fd** special file names (except the generic special files **/dev/rfd0** and **/dev/rfd1**) contain suffixes that dictate how a diskette is to be treated. These special file names have a format of *PrefixXY*, where *Prefix, X* and *Y* have the following meanings:

Prefix — Special file type. Possible values are **fd** and **rfd**, where the **r** indicates raw access to the special file.

X — Drive number indicator. Possible values of **0** and **1** indicate drives 0 and 1, respectively.

Y — Diskette format indicator. Possible values depend on the type of diskette being used. Either a single character or a decimal point with following characters are allowed. Possible values are **h** (high-density 3.5-inch drive), **l** (low-density 3.5-inch drive), **.15** (15 sectors per track on a 5.25-inch drive) and **.9** (9 sectors per track on a 5.25-inch drive).

3.5-Inch Diskette Special Files

Eight different special files are available for use with the 3.5-inch diskette drive only. The default diskette type assumed for a 1.44 M-byte drive is a double-sided, 80-cylinder, 18 sectors-per-track diskette.

An **h** as the suffix of the special file name (for example, **/dev/rfd0h**) forces a diskette to be treated as a double-sided, 80-cylinder, 18 sectors-per-track diskette. An **l** as the suffix of the special file name (for example, **/dev/rfd1l**)

forces a diskette to be treated as a double-sided, 80-cylinder, 9 sectors-per-track diskette.

5.25 Inch Diskette Special Files

Eight different special files are available for use with the 5.25-inch diskette drive only. The default diskette type assumed for a diskette in a 1.2 M-byte drive is a double-sided, 80-cylinder, 15 sectors-per-track diskette.

A **.15** as the suffix of the special file name (for example, **/dev/rfd0.15**) forces a diskette to be treated as a double-sided, 80-cylinder, 15 sectors-per-track diskette. A **.9** as the suffix of the special file name (eg, **/dev/rfd0.9**) forces a diskette to be treated as a double-sided, 40-cylinder, 9 sectors-per-track diskette.

Usage Considerations

The open and close Subroutines

Only one process at time can issue an **open** subroutine call to gain access to a particular drive. However, all children created by a process that successfully opens a diskette drive will inherit the open diskette drive.

The read and write Subroutines

No special considerations.

The ioctl Subroutine

The possible **ioctl** operations and their descriptions are:

IOCINFO Returns a **devinfo** structure (defined in the **<sys/devinfo.h>** header file) that describes the device.

FDIOCSINFO Sets the device driver diskette characteristics to the values passed in the **fdinfo** structure, as defined in the **<sys/fd.h>** header file.

FDIOCSINFO Gets the device driver diskette characteristics and returns the values in the **fdinfo** structure, as defined in the **<sys/fd.h>** header file.

FDIOCFORMAT

Formats a diskette track. The diskette is formatted using data passed in an array of bytes. The length of this array is 4 times the number of sectors per track on the diskette. The reason for this is that 4 bytes of data must be passed in for every sector on the track. The 4 bytes contain, in this order, the cylinder number, the side number (0 or 1), the sector number, and the number of bytes per sector. This pattern must be repeated for every sector on the track.

The diskette characteristics used during formatting are whatever values are in the device driver when it receives the format command. These characteristics need to be set to the desired values prior to issuing the format command. This can be accomplished in three different ways:

- Open the diskette driver using one of the format-specific special files. As a result, the diskette characteristics for the driver will be those of the diskette indicated by the special file.

- Open the diskette driver using one of the generic special files. In this case, the diskette characteristics will be the default characteristics for that driver.
- Set the characteristics explicitly using the FDIOCSINFO **ioctl** operation.

For formatting, the diskette driver should be opened with the O_NDELAY flag set. Otherwise, the driver will attempt to determine the type of diskette in the drive, causing the open to fail.

Implementation Specifics

The **fd** special file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/fd0, **/dev/fd1**, **/dev/rfd0**, **/dev/rfd1**

/dev/fd0h, **/dev/fd0l**, **/dev/fd0.9**, **/dev/fd0.15**
/dev/fd1h, **/dev/fd1l**, **/dev/fd1.9**, **/dev/fd1.15**
/dev/rfd0h, **/dev/rfd0l**, **/dev/rfd0.9**, **/dev/rfd0.15**
/dev/rfd1h, **/dev/rfd1l**, **/dev/rfd1.9**, **/dev/rfd1.15**

Related Information

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **lseek** subroutine, **ioctl** subroutine.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files, Understanding Device Driver Classes, Understanding Block I/O Device Drivers, Understanding Character I/O Device Drivers, and Providing Raw I/O Support in *Kernel Extensions and Device Support Programming Concepts.*

hft Special File

Purpose

Implements a high function terminal (HFT) device.

Description

The **hft** device driver file is the application interface to the HFT subsystem (HFTSS). The HFT device driver supports the concept of a *virtual terminal*. Virtual terminals are logically independent of each other but share physical resources over time. The virtual terminal that can accept physical input at a given time is called the *active virtual terminal*. 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.

The physical components that virtual terminals support include display devices, keyboards, locators, valuators, lighted programmable function keys (LPFKs), and sound generators. The HFT device driver is a collection of terminal-related device drivers that service the various physical components of a terminal device. Thus the HFT device driver, as a virtual terminal driver, provides a unified interface for accessing the separate components of a virtual terminal.

The **hft** file is the kernel-level support for virtual terminals. Because 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 **ioctl** subroutines to the HFT driver. A number of **ioctl** functions are provided to allow access to the many HFT operations.

HFT Physical and Virtual Terminals

A work station can have several virtual terminals open at the same time. Each of these virtual terminals is logically independent of all other virtual terminals but shares the same set of physical resources.

The HFTSS supports as many as 32 virtual terminals at one time. The virtual terminals can be opened from as many as 4 physical terminals. In a non-windowing environment, only one virtual terminal is visible on the screen at any time. While only one virtual terminal can accept input at any given time (the active virtual terminal), as many as 4 virtual terminals, each on a separate physical display device, can process output concurrently.

Each time the **hft** special file is opened, a new HFT virtual terminal is created and opened. To reopen an existing virtual terminal, open the /**dev/hft/***n* special file, where *n* represents the number of an open driver channel. The channel number can be determined with the HFGETID **ioctl** operation.

HFT Initial State

The HFT supplies default values for the following virtual terminal facilities:

- Keyboard-to-character mapping, which provides a mechanism that takes input from the keyboard device driver and translates it to a displayable or ASCII control character.
- Character-to-display mapping, which provides the default font used to echo characters to the screen.
- Echo-break specification, which provides the default map used by the display device to echo characters to the screen and to break if there are none.

- Tab rack, which provides default tabs at the first, last, and every eighth position in the presentation space.
- Protocol mode flags, which translate all keyboard input without sending it to the operating system.

These default values are used until a redefinition is received from the application.

HFT Virtual Terminal Modes

The virtual terminal provides a model of a single terminal that can be in either the keyboard send-receive (KSR) mode or the monitor mode (MOM). The KSR mode emulates an ASCII terminal using an ASCII data stream. The monitor mode allows applications to have a direct output path to the display hardware and a shortened path for input. 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 that expect an ASCII terminal.

HFT Operations

The **hft.h** header file contains the structure definitions for an HFT device. The file structures in the **hft.h** file pertain to the following specific HFT operations:

- query ioctl operations
- special ioctl operations
- read operations
- write operations
- keyboard send-receive mode (KSR) operations
- monitor mode (MOM) operations.

The **/usr/lib/samples/hft** directory contains sample programs that use the HFT virtual terminal subsystem.

For general information about special files, refer to the Special Files Overview. For genereral information about the HFT subsystem, refer to the HFT Subsystem Conceptual Introduction and related articles in *Communication Concepts and Procedures*.

HFT Query ioctl Operations

Several structures in the **hft.h** header file support **ioctl** operations, which provide access to sophisticated features of the high function terminal (HFT) subsystem.

The following query **ioctl** operations are supported to request information about the HFT subsystem:

- Get virtual terminal ID
- Query I/O Error
- Query device
- Query
- Query screen manager.

Several other specialized **ioctl** operations are also available. These are discussed in HFT Special ioctl Operations.

Get Virtual Terminal ID (HFTGETID)

The HFTGETID operation gets identification information for the current HFT virtual terminal. The **hftgetid** structure contains the following fields:

hf_dev	The major and minor device number of the HFT subsystem.
hf_pgrp	The process group ID of the terminal group leader.
hf_chan	The channel number.

Query I/O Error (HFQERROR)

The HFQERROR operation returns detailed device error code information. If an I/O operation or other subroutine to the HFT is unsuccessful due to an HFT subsystem error, a nonzero value is returned and the **errno** variable is set to EIO. The calling program can get a more detailed device error code by using the **ioctl** subroutine to issue an HFQERROR operation, as follows:

```
int ioctl (FileDescriptor,HFQERROR, Argument)
        int FileDescriptor;
        int *Argument
```

The argument from the HFQERROR **ioctl** operation is either a value of 0 (indicating that the last I/O operation was successful) or the error code for the last HFT I/O operation.

Query Device (HFTQDEV)

The HFTQDEV operation obtains information about the devices associated with the HFT subsystem and stores the information in an **hftqdresp** structure. The **hftqdresp** structure contains the following fields:

hf_vtid	Virtual terminal ID number.
hf_dev	Major and minor number of the HFT device driver.
hf_numdisp	Number of display devices actually installed.
hf_kbddev	Major and minor number of the keyboard device driver.
hf_mousedev	Major and minor number of the mouse device driver.
hf_snddev	Major and minor number of the sound device driver.
hf_tabletdev	Major and minor number of the tablet device driver.
hf_dialsdev	Major and minor number of the dials device driver.
hf_lpfkdev	Major and minor number of the lighted programmable function keys (LPFKs) device driver.
hf_dispdev[*i*]	Major and minor number of the display monitors attached to the HFT subsystem.
hf_defdisp	Major and minor number of the default display monitor.

Query (HFQUERY)

The HFQUERY operation gets information about the current virtual terminal. The hf_cmd and hf_cmdlen fields in the **hfquery** structure describe a buffer containing the command. The hf_resp and hf_resplen fields describe a buffer large enough to hold the expected response.

Structures for each the following query commands are defined in the **hft.h** header file.

- Query Physical Display IDs
- Query Physical Device
- Query Mouse, Query Tablet
- Query LPFKs
- Query Dials
- Query Presentation Space
- Query Software Keyboards
- Query HFT Device
- Query Keyboard Status
- Query Retract Device ID.

Query Physical Display IDs Command

The Query Physical Display IDs command returns the number of display devices and the ID of each display device attached to the system. The command uses the **hfqdevidc** structure, which contains the following fields and values:

hf_intro.hf_typehi HFQDEVIDCH.

hf_intro.hf_typelo HFQDEVIDCL.

The returned information is in the form of an **hfqdevidr** structure, which contains the following fields and values:

hf_intro.hf_typehi HFQDEVIDRH.

hf_intro.hf_typelo HFQDEVIDRL.

hf_numdev The number of display devices for which data is reported.

The following two fields are repeated for each display device:

hf_devid Physical device ID. The following values are possible:

0x0421*mmnn*	IBM Color Graphics Display Adapter
0x0422*mmnn*	IBM Grayscale Graphics Display Adapter
0x0425*mmnn*	IBM High Performance 3D Color Graphics Processor
0x0429*mmnn*	IBM High Speed 3D Graphics Accelerator

mm indicates the state of the adapter: 0x00 indicates that the adapter is totally functional; other values indicate that the adapter is present but not totally functional.

nn differentiates between multiple instances of the same adapter type (0x00 to 0x03).

hf_class — Display class (0x44).

Query Physical Device Command

This command returns information about display and locator devices. The **hfqphdevc** structure, which is used to issue this command, contains the following fields and values:

hf_intro.hf_typehi — HFQPDEVCH.

hf_intro.hf_typelo — HFQPDEVCL.

hf_phdevid — Physical display ID that is queried and returned as the hf_devid field in the Query Physical Display IDs command response. Any display device ID can be queried, or this field can be set to a value of 0 to query the display device to which the currently active terminal is attached.

The display and locator device information that this command returns fills the response buffer in the form of an **hfqphdevr** structure. The fields and values of the **hfqphdevr** structure are described in the following paragraphs.

Physical Device Information VTD Header:

hf_intro.hf_typehi — HFQPDEVRH.

hf_intro.hf_typelo — HFQPDEVRL.

Physical Locator Information:

hf_scale[0] — The mouse scale factor (millimeters per 100 counts):

HFMSCALE1	Scaling for mouse is 1:1.
HFMSCALE2	Scaling for mouse is 2:1.

hf_scale[1] — The tablet conversion:

HFTCONVE	English units.
HFTCONVM	Metric units.

hf_locattr[1] — Locator attributes. Indicates the presence of a mouse or tablet:

HFLOCREL	If set, indicates that a mouse is attached.
HFLOCABS	If set, indicates that a tablet is attached.

Physical Display Device Information:

hf_attrib[0] — Display device attributes:

HFISAPA	All points addressable (APA) display device.
HFHASBLINK	Blink function allowed.

All other values are reserved.

hf_attrib[2]	Display device attributes: HFHASCOLOR Color allowed. All other values are reserved.
hf_attrib[3]	Display device attributes: HFCHGPALET The display adapter color palette can be changed. All other values are reserved.
hf_pwidth	Displayable width of the physical screen, expressed in pels (pixels).
hf_pheight	Displayable height of the physical screen, expressed in pels.
hf_mwidth	Displayable width in millimeters.
hf_mheight	Displayable height in millimeters.
hf_bperpel	Bits per pel (1, 2, or 4).
hf_phdevid	Physical display ID.

Physical Display Font Information:

hf_numfont	Number of fonts available to this display device.

The following fields are repeated for each available font:

hf_fontid	Physical font ID.
hf_fontname	Name of the font (usually a file name).
hf_fontweight	Weight of the font (such as bold, semibold, and book).
hf_fontstyle	Physical font style (such as Roman and italics).
hf_fontencode	Code page this font renders (such as PC850).
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:

hf_numcolor	Total number of colors possible.
hf_numactive	Number of colors that can be active at any one time (16 maximum).
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 from 0 through hf_numactive – 1, and is repeated for each of the currently active colors. Each occurrence contains a hardware-dependent color specification.

Query Mouse Command, Query Tablet Command

This command returns device information about a mouse or tablet input device connected to a virtual terminal. The **hfqgraphdev** structure, which consists of the following fields, is used to issue this command.

hf_intro.hf_typehi	HFQMOUSECH or HFQMTABLETCH.
hf_intro.hf_typelo	HFQMOUSECL or HFQMTABLETCL.

This command fills the response buffer with information about the locator device in the form of an **hfqlocr** structure. The **hfqlocr** structure contains the following fields and values:

hf_intro.hf_typehi	HFQMOUSERH or HFQMTABLETRH.
hf_intro.hf_typelo	HFQMOUSERL or HFQMTABLETRL.
hf_resolution	The resolution of the mouse or tablet. This is a 4-byte value expressed as millimeters per 100 count.
hf_devinfo[0]	Locator attributes:
	HFLOCABS — If set, these are the absolute coordinates for a tablet.
	HFLOCREL — If set, these are the relative coordinates for a mouse.
	HFLOCUNKNOWN — This is an unknown tablet sensor type.
	HFLOCSTYLUS — The tablet has a stylus sensor.
	HFLOCPUCK — The tablet has a puck sensor.
hf_horzmax_cnt	The tablet horizontal maximum count (a 2-byte value).
hf_vertmax_cnt	The tablet vertical maximum count (a 2-byte value).
hf_horzdead_zone	The tablet horizontal dead zone or horizontal threshold of the mouse.
hf_vertdead_zone	The tablet vertical dead zone or vertical threshold of the mouse.
hf_sample_rate	The sample rate of the mouse or tablet.
hf_origin	The tablet origin, which can be the HFMIDDLE value (middle of tablet), or the HFLOWERL value (lower left corner of tablet).
hf_scale [0]	The mouse scaling factor, as follows:
	HFMSCALE1 — Scaling for the mouse is 1:1.
	HFMSCALE2 — Scaling for the mouse is 2:1.
hf_scale [1]	The tablet conversion, as follows:
	HFTCONVE — English units.
	HFTCONVM — Metric units.

Query Lighted Programmable Function Keys (LPFKs) Command

This command returns device information about the LPFKs. The **hfqgraphdev** structure, which contains the following fields and values, is used to issue this command.

hf_intro.hf_typehi	HFQLPFKSCH.
hf_intro.hf_typelo	HFQLPFKSCL.

This command fills the response buffer with LPFK information in the form of an **hfdial_lpfk** structure. The **hfdial_lpfk** structure contains the following fields and values:

hf_intro.hf_typehi	HFQLPFKSRH.
hf_intro.hf_typelo	HFQLPFKSRL.
hf_data1.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 a value of 1 indicate enabled LPFKs; bits set to a value of 0 indicate disabled LPFKs.

Query Dials Command

This command returns device information about the dials. The **hfqgraphdev** structure, which contains the following fields and values, is used to issue this command.

hf_intro.hf_typehi	HFQDIALSCH.
hf_intro.hf_typelo	HFQDIALSCL.

This command fills the response buffer with dial information in the form of an **hfdial_lpfk** structure and contains the following fields and values:

hf_intro.hf_typehi	HFQDIALSRH.
hf_intro.hf_typelo	HFQDIALSRL.
hf_data1.hf_numdials	Number of dials on the device.
hf_data2.granularity	An array of sixteen 1-byte values giving the number of events per full 360-degree revolution (the granularity) of each dial. The values in the array (0 through 8) represent powers of 2 (0=off, 1=2 events per 360-degree revolution, 2=4 events, 4=16 events, and so forth).

Query Presentation Space Command

This command returns an ASCII data stream image of the current display screen. All or part of the screen can be queried. Attribute and character set information on the queried block are returned. This query is valid only in keyboard send-receive (KSR) mode.

The **hfqpresc** structure, which is used to issue this command, contains the following fields and values:

hf_intro.hf_typehi	HFQPRESCH.
hf_intro.hf_typelo	HFQPRESCL.

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).

This command fills the response buffer with attribute and character set information in the form of an **hfqpresr** structure. The **hfqpresr** structure contains the following fields and values:

hf_intro.hf_typehi	HFQPRESRH.
hf_intro.hf_typelo	HFQPRESRL.

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 Set Graphic Rendition (SGR) and Set G0 or G1 Character Set (SG0) control sequences. A linefeed control is returned after the last character code in each line of the queried block

Note: The returned attributes might be only a subset of the original attributes that were output to the display device. Only those attributes that are actually supported by the physical device are returned. Possible attributes include boldface, underscored, blinking, reverse image, or invisible characters. Font selection and foreground and background color selection are also supported.

Query Software Keyboards Command

This command returns status information about the software keyboard maps that are loaded in the HFT. The **hfqswbc** structure, which is used to issue this command, contains the following fields and values:

hf_intro.hf_typehi	HFQSWKBCH.
hf_intro.hf_typelo	HFQSWKBCL.

This command fills the response buffer with information about the available software maps. This information is in the form of an **hfqswbr** structure, which contains the following fields and values:

hf_intro.hf_typehi	HFQSWKBRH.
hf_intro.hf_typelo	HFQSWKBRL.
hf_numkbds	The number of software keyboard maps loaded.

The following fields are repeated for each available software keyboard map:

hf_disp_set	The display symbol set of this software keyboard map (for example, PC850).
hf_kbdname	A descriptive name for this software keyboard map.
hf_kbdid	The unique identifier of this software keyboard map.

Query HFT Device Command

This command returns information about the HFT device. The **hfqhftc** structure, which is used to issue this command, contains the following fields and values:

hf_intro.hf_typehi	HFQHFTCH
hf_intro.hf_typelo	HFQHFTCL.

This command fills the response buffer with HFT device information in the form of an **hfqhftr** structure. The **hfqhftr** structure contains the following fields and values:

hf_intro.hf_typehi	HFQHFTRH.
hf_intro.hf_typelo	HFQHFTRL.
hf_phdevid	Physical display ID (the same as returned by the Query Physical Display IDs command).
hf_phrow	The number of character rows in the presentation space, based on the current font.
hf_phcol	The number of character columns in the presentation space, based on the current font.
hf_phcolor	The number of different colors supported by the display device.
hf_phfont	The number of fonts defined in the system.
hf_phkbdid	Physical keyboard ID:
	HF101KBD 101-key keyboard.
	HF102KBD 102-key keyboard.
	HF106KBD 106-key keyboard.

Query Keyboard Status Command

This command returns status information about the state of a keyboard. The **hfqkbsc** structure, which contains the following fields and values, is used to issue this command.

hf_intro.hf_typehi	HFQKBSCH.
hf_intro.hf_typelo	HFQKBSCL.

This command fills the response buffer with keyboard status information in the form of an **hfqkbsr** structure. The **hfqkbsr** structure contains the following fields and values:

hf_intro.hf_typehi	HFQKBSRH.
hf_intro.hf_typelo	HFQKBSRL.
hf_kbstat[3]	Defines the keyboard state:
	HFALPHAKBD Keyboard is in alpha (roma-ji) state.
	HFKATAKBD Keyboard is in katakana state.
	HFHIRAKBD Keyboard is in hiragana state.

Query Retract Device ID Command

This command returns the display device ID and virtual terminal ID of the virtual terminal being retracted. The **hfqretractc** structure, which contains the following fields and values, is used to issue this command.

hf_intro.hf_typehi	HFQRETRACTCH.
hf_intro.hf_typelo	HFQRETRACTCL.

This command fills the response buffer with the display device ID and virtual terminal ID of the virtual terminal being retracted. The information is in the form of an **hfqretractr** structure, which contains the following fields and values:

hf_intro.hf_typehi	HFQRETRACTRH.
hf_intro.hf_typelo	HFQRETRACTRL.
hf_devid	Physical display device.
hf_vtid	Virtual terminal ID.

Query Screen Manager (HFTQSMGR)

The HFTQSMGR operation queries the screen manager ring for each virtual terminal, returning the multiplex number (vtid) and state of each. The file descriptor can be associated with any virtual terminal in the HFT subsystem. The **hfbuf** structure is used when issuing this command. The hf_bufp field in the **hfbuf** structure points to an **hftqstat** structure, and the hf_buflen field in the **hfbuf** structure indicates the length of the **hftqstat** structure. The **hftqstat** structure contains the following fields and values:

hf_numvts	Specifies the number of virtual terminals.

The following fields are repeated for each virtual terminal in the screen manager ring:

hf_vtid	Specifies the ID number of the virtual terminal.
hf_vtstate	Specifies the status of the virtual terminal:

HFVTHIDDEN	Virtual terminal is hidden.
HFVTACTIVE	Virtual terminal is active.
HFVTCOMMAND	Virtual terminal is the command terminal.
HFVTTRUSTED	Virtual terminal is a trusted terminal.
HFVTNOHOTKEY	Virtual terminal cannot be activated by a hot key.

Also refer to the descriptions of the **hft.h** structures for special ioctl operations, read operations, general write operations, KSR write operations, and MOM write operations.

HFT Special ioctl Operations

Several structures in the **hft.h** header file support **ioctl** operations, which provide access to sophisticated features of the high function terminal (HFT) subsystem.

The following special **ioctl** operations are supported to perform other specialized HFT functions:

- Reconfigure
- Set echo and break maps
- Set keyboard map
- Enable or disable sound signal
- Enter or exit monitor mode
- Control screen manager
- Enable software keyboard
- Change locator.

Several query **ioctl** operations are also available. These are discussed in the following section:

HFT Query ioctl Operations

Reconfigure (HFRCONF)

The HFRCONF operation allows a user program to reconfigure the HFT subsystem. The command changes the configuration of the HFT subsystem defaults. The **hfrconf** structure contains two fields: the hf_op field contains the requested operation, and the hf_obj field identifies the object of the operation. The following list describes the valid operations for the hf_op field. These reconfigure operations, with the exception of those followed by an * (asterisk), take effect only for virtual terminals that are 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.

HFTADDFONT	Adds the font indicated in the hf_obj field. The hf_obj field contains the font file descriptor from an **open** subroutine.
HFCHGKBDRATE*	Changes the automatic repeat mode rate of the keyboard. The hf_obj field indicates the rate at which the keyboard takes input and sends it to the HFT. Valid automatic repeat mode rate values are from 2 through 30 characters per second (char/sec) and can be incremented in 1-char/sec units. The default value is 11 char/sec.
HFCHGKBDDEL*	Changes the automatic repeat mode delay of the keyboard. The hf_obj field indicates how long a key must be pressed before the keyboard goes into the automatic repeat mode. Valid delay values are between 250 and 1000 milliseconds (ms) and can be incremented in 250-ms units. The default value is 500 ms.

HFCHGCLICK*	Turns the keyboard click mechanism on or off. The hf_obj field indicates whether the keyboard produces a sound when a key is pressed. Sound is suppressed when the value of the hf_obj field equals HFCLICKOFF. Sound is produced when the value of the hf_obj field equals HFCLICKON. The default value is keyboard click off (HFCLICKOFF).
HFCHGVOLUME*	Sets the sound volume level. The hf_obj field indicates the volume of sounds produced by the speaker. For the standard speaker, valid values are 0 (sound off), 1 (low volume), 2 (medium volume), and 3 (high volume). The default value for the speaker is 2.
HFTECHOMAP	Replaces the echo map. The hf_obj field contains the new echo map file descriptor from an **open** subroutine.
HFTBREAKMAP	Replaces the break map. The hf_obj field contains the new break map file descriptor from an **open** subroutine.
HFTDEFAULT	Replaces miscellaneous default values. The hf_obj field contains the new miscellaneous defaults file descriptor from an **open** subroutine.
HFSETDD	Changes the default display. The hf_obj field contains the physical display identifier returned by an HFQUERY Physical Display IDs **ioctl** operation.
HFADDSWKBD	Adds a new software keyboard map to the system. The hf_obj field contains the new software keyboard map file descriptor from an **open** subroutine.
HFCHGSWKBD	Changes the default software keyboard map. The hf_obj field contains the software keyboard map identifier returned by an HFQUERY Software Keyboards **ioctl** operation.

Set Echo (HFTSECHO) and Break Maps (HFTSBREAK)

The **echo_map** and **break_map** structures are used for these operations. These structures consist of 16 consecutive words of storage aligned on a word boundary. The words are numbered from 0 through 15; word 0 consists of bits 0 through 31, word 1 consists of bits 32 through 63, and so forth. The hf_buflen field contains the length of the array in bytes.

Set Keyboard Map (HFSKBD)

The HFSKBD operation sets the keyboard map. Most keys on the keyboard can be remapped, changing the character or control sequence that each key generates when it is pressed. The hf_bufp field in the **hfbuf** structure points to a **hfkeymap** structure. The hf_buflen field in the **hfbuf** structure contains the length of the **hfkeymap** structure.

The **hfkeymap** structure can remap one or more keys. The number of keys to remap is specified in the hf_nkeys field. One **hfkey** structure for each key specified in the hf_nkeys field follows. The HFNKEYS constant, which is used as the dimension for the hfkey array, has a default value of 1, which allows one key to be remapped. To change the HFNKEYS constant, set its value in a **#define** statement that comes before the **#include <sys/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.
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 non-spacing character to a key.
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. (Use the left Alt key.)
HFSHFALTGR	Specifies the Alt Gr (Alternate Graphics) state. (Use the right Alt key.)
HFCAPSL	Specifies whether the Caps Lock state affects the key. If set, when Caps Lock mode is on, the base state of a key functions as the Shift state, and the Shift state 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 the hf_kstate field of the **hfkey** structure:

HFMAPCHAR and HFMAPNONSP

hf_char	Specifies a character.

HFMAPSTR

hf_kstrl	Specifies the length of a string in bytes minus 1. This is immediately followed by the string.

Note: When using an **hfkeymap** structure, only the last key specified in the hfkey array can be assigned a string value. However, the HFT subsystem allows any number of keys to be assigned string values.

HFMAPFUNC

hf_keyidh Specifies the high-order byte of the function ID.

hf_keyidl Specifies the low-order byte of the function ID.

The function IDs that can be assigned for the keys are:

ID	Name	Description
0x0000–0x00FE	PFK	Issues the program function (PF) 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 to the beginning of the field.
0x0106	CHT	Moves the application cursor to the next horizontal tab stop.
0x0107	CVT	Moves the application cursor down one vertical tab stop.
0x0108	CUP	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	EL	Erases to the end of the line.
0x0155	EF	Erases to the next tab stop.
0x0156	CLEAR	Erases all characters from the presentation space.
0x0157	RIS	Restores the initial state of the virtual terminal.
0x0162	RI	Performs one line reverse index control.
0x0163	IND	Performs one line index control.
0x01FF	IGNORE	Sends no information when the key is pressed.

Enable and Disable Sound Signal (HFESOUND and HFDSOUND)

The HFESOUND operation informs the virtual terminal of the intent to use sound, enabling the routing of the sound response signal. This command is issued using the **hfsmon** structure, which includes the following field:

hf_momflags	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.

This signal is used to let an application know that a sound has been queued for the speaker device. It can be used for pacing and other similar functions.

The HFDSOUND operation disables the sound signal by informing the virtual terminal of the intent to discontinue the use of sound signals, in which case, sound signals are not sent. There are no structures required to disable sound signals.

Enter and Exit Monitor Mode (HFSMON and HFCMON)

The HFSMON operation requests monitor mode. Monitor mode provides a program with direct control of the screen and keyboard. This command is issued using the **hfsmon** structure, which includes the following field:

hf_momflags	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.

The HFCMON operation releases monitor mode. There are no structures required to clear monitor mode.

Control Screen Manager (HFTCSMGR)

The HFTCSMGR operation requests the screen manager to manipulate the status of virtual terminals. The file descriptor can be associated with any virtual terminal in the HFT subsystem. This operation is issued using the **hftsmgrcmd** structure, which contains the following fields:

hf_vtid	The multiplex number of the virtual terminal.	
hf_vsid	Reserved.	
hf_cmd	Contains one of the following screen manager commands:	
	SMACT	Activates the virtual terminal.
	SMHIDE	Hides the virtual terminal. This command marks the terminal specified in the hf_vtid field so that the screen manager does not activate it. If this terminal is at the head of the ring, another virtual terminal is made active when this terminal is hidden.

SMSCMD	Sets the command virtual terminal. This command designates the terminal that is specified in the hf_vtid field as the command virtual terminal. The command virtual terminal is activated by pressing the command window hot key.
SMUNHIDE	Unmarks the hidden terminal. This command restores to the screen manager ring the hidden terminal that is specified in the hf_vtid field. The virtual terminal becomes visible and active if it is at the head of the ring when this command is issued. This command is not valid if either the active terminal or the object terminal is trusted.
SMCVTEN	Causes the command virtual terminal to be activated when both of the outermost mouse buttons are pressed at the same time, the Ctrl–Alt key sequence is pressed, or button 4 on the tablet is pressed. This is the default setting.
SMCVTDI	Disables command terminal hot keying.
SMNOHOTKEY	Marks the virtual terminal so that it cannot be activated by a hot key sequence.

Enable Software Keyboard (HFESWKBD)

This command changes the software keyboard map currently used by a virtual terminal to another keyboard map. The new keyboard map must currently be defined in the system. If it is not, use the HFADDSWKBD operation of the reconfigure (HFRCONF) **ioctl** to add it. The query (HFQUERY) **ioctl** shows the available keyboard maps. The HFESWKBD operation invokes the enable software keyboard operation. The argument for the operation points to the hf_kbdid field, which contains the unique identifier of the software keyboard map to be enabled. The unique identifier is returned from the hf_kbdid field of the Query Software Keyboards operation.

Change Locator (HFCHGLOC)

This command allows the user to change the mouse sample rate, mouse resolution, mouse thresholds, mouse scaling, tablet sample rate, tablet resolution, tablet dead zone, tablet conversion, or tablet origin. The change locator operation is issued using the HFCHGLOC command. The argument to the command points to the **hfchgloc** structure, which contains the following fields and values:

hf_cmd — The command that is to be issued to the device driver. The following are valid values:

HFMRATE	Set mouse sample rate.
HFMRES	Set mouse resolution.
HFMTHRESH	Set mouse thresholds.
HFMSCALE	Set mouse scaling.
HFTRATE	Set tablet sample rate.
HFTRES	Set tablet resolution.

HFTDZONES	Set tablet dead zones.
HFTORIGIN	Set tablet origin.
HFTCONV	Set tablet conversion.

hf_value1 The value of the sample rate, resolution, origin, scaling, conversion, horizontal threshold, or horizontal dead zone.

Valid mouse sample rates are 10, 20, 40, 60, 80, 100, or 200 samples per second.

Valid tablet sample rates are 1 to 100 samples per second.

Valid mouse resolutions are 0, 1, 4, or 8 counts per millimeter.

Valid tablet resolutions are 0 through 1279 counts per millimeter.

Valid mouse horizontal thresholds are values from 0 to 32767.

Valid tablet horizontal dead zones are values from 0 to 32767.

Valid origin indicators are:

HFMIDDLE	Set the tablet origin to the middle of the tablet.
HFLOWERL	Set the tablet origin to the lower-left corner of the tablet.

Valid mouse scaling indicators are:

HFMSCALE1	1:1.
HFMSCALE2	2:1.

Valid tablet conversion values are:

HFTCONVE	English.
HFTCONVM	Metric.

long hf_value2 The valid mouse vertical threshold or tablet vertical dead zone. This can be a value from 0 to 32767.

Also refer to the descriptions of the **hft.h** structures for query ioctl operations, read operations, general write operations, KSR write operations, and MOM write operations.

HFT read Operations

Several structures in the **hft.h** header file support input to the high function terminal (HFT) subsystem from other devices. Data from devices other than the keyboard is passed back from the **read** subroutine in the form of special control sequences, which are defined in the **hft.h** header file. These control sequences include:

- Untranslated keyboard input
- Input device report.

Untranslated Keyboard Input

If keyboard input is received when the HFXLATKBD protocol mode is turned off, an untranslated keystroke is returned. The key position identifies the physical key pressed. The key status bits indicate Alternate (Alt), Alternate Graphics (Alt–Gr), Control (Ctrl), Shift, Caps Lock, and Number Lock key states. The scan code and make and break keys are dependent upon hardware and require knowledge of the physical keyboard in use. The **hfunxlate** structure, used for this operation, contains the following fields:

hf_status[0]	Status:	
	HFUXSHIFT	A Shift key is pressed.
	HFUXCTRL	The Ctrl key is pressed.
	HFUXALT	An Alt key is pressed.
	HFUXCAPS	The Caps Lock mode is in effect.
	HFUXNUM	The Number Lock mode is in effect.
	HFUXSCROLL	The Scroll Lock mode is in effect.
	HFUXMAKE	If set, a key has been pressed. If not set, a key has been released.
hf_status[1]	Status:	
	HFUXRPT	Automatic repeat state.
	HFUXLSH	Left Shift state.
	HFUXRSH	Right Shift state.
	HFUXLALT	Left Alt–Shift state
	HFUXRALT	Right Alt–Shift state. (Alt–Gr for 102-key keyboards)
hf_seconds	The time of the report in whole seconds since system startup.	
hf_sixtyths	The fractional part of the keyboard report time stamp, in 1/60th seconds.	

Input Device Report

The **hflocator** structure is used when reporting input data from a device. This structure contains fields for the following input devices:

- Mouse
- Tablet
- Lighted programmable function keys
- Valuator dials.

Mouse Report

hf_deltax	The x delta, a twos complement signed integer that holds the relative x delta accumulations in counts of 0.25 millimeters of the mouse movement. This information is sent to the virtual terminal to indicate horizontal movement since the last mouse movement.
hf_deltay	The y delta, a two's complement signed integer that holds the relative y delta accumulations in counts of 0.25 millimeters of the mouse movement. This information is sent to the virtual terminal to indicate vertical movement since the last mouse movement.
hf_seconds	The time of the mouse report in whole seconds since system startup.
hf_sixtyths	The fractional part of the mouse report time stamp, in 1/60th seconds.

hf_buttons — The status of the mouse buttons. This information is sent to the virtual terminal to indicate a change in the status of the buttons since the last mouse movement, as follows:

HFBUTTON1	Button 1 (rightmost) has been pressed.
HFBUTTON2	Button 2 has been pressed.
HFBUTTON3	Button 3 has been pressed.
HFBUTTON4	Button 4 (leftmost) has been pressed.

hf_stype — The source of the input. (A value of 0 indicates that the input is from the mouse.)

Tablet Report

hf_deltax — The absolute x coordinate of the tablet sensor.

hf_deltay — The absolute y coordinate of the tablet sensor.

hf_seconds — The time of the tablet report in whole seconds since system startup.

hf_sixtyths — The fractional part of the tablet report time stamp, in 1/60th seconds.

hf_buttons — The status of the tablet buttons. This information is sent to the virtual terminal to indicate a change in the status of the buttons since the last tablet movement, as follows:

HFBUTTON1	Button 1 (rightmost) has been pressed.
HFBUTTON2	Button 2 has been pressed.
HFBUTTON3	Button 3 has been pressed.
HFBUTTON4	Button 4 (leftmost) has been pressed.
HFBUTTON_STAT	The presence of a stylus or puck is detected over the tablet.

hf_stype — The source of the input. (A value of 1 indicates that the input is from a tablet.)

LPFK Report

hf_deltax — The lighted programmable function key (LPFK) number.

hf_deltay — Reserved.

hf_seconds — The time of the report in whole seconds since system startup.

hf_sixtyths — The fractional part of the locator report time stamp, in 1/60th seconds.

hf_buttons — Not used.

hf_stype — The source of the input. (A value of 2 indicates that the input is from LPFKs.)

Valuator Dial Report

hf_deltax — The dial number.

hf_deltay — The dial value delta. This is a signed integer value in the dial units of granularity, as defined in the **hfdial_lpfk** structure.

hf_seconds — The time of the report in whole seconds since system startup.

hf_sixtyths	The fractional part of the locator report time stamp, in 1/60th seconds.
hf_buttons	Not used.
hf_stype	The source of the input. (A value of 3 indicates that the input is from dials.)

Also refer to the descriptions of the **hft.h** structures for query ioctl operations, special ioctl operations, general write operations, KSR write operations, and MOM write operations.

HFT General write Operations

ASCII data can be sent to the virtual terminal using the **write** subroutine along with data of any length. In addition, virtual terminal control structures can be sent to the virtual terminal using the **write** subroutine.

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 prefix is followed by 4 bytes and an operation type code. The data that follows this structure depends on the type of control.

The significant fields in the **hfintro** structure are:

hf_len	The total number of bytes in the header and associated data, not including the 3-character VTD control sequence.
hf_typehi	The high-order byte of the information type code.
hf_typelo	The low-order byte of the information type code.

Because the **hfintro** structure is an odd number of bytes in length, it is designated as the hf_intro[HFINTROSZ] character array 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 also referred to as hf_intro.hf_typehi and hf_intro.hf_typelo.

All reserved and unused fields must be set to a value of 0. You can set the entire structure to a value of 0 and then fill in the appropriate fields.

Several structures in the **usr/include/sys/hft.h** header file support the following high function terminal (HFT) write operations:

- Set Protocol Modes
- Set Keyboard LEDs
- Set LPFKs and Set Dial Granularities
- Write Sound
- Cancel Sound
- Change Physical Display.

Set Protocol Modes

Protocol modes determine how the virtual terminal interprets, translates, and returns 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, the corresponding bit in the hf_value field indicates the new setting for the mode. The mode bits are set to the default value when the virtual terminal is opened. These defaults can be changed for subsequently opened HFTs with the HFRCONF operation.

The **hfprotocol** structure, which gives the protocol definitions, contains the following fields:

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		Valid only in the KSR mode. Specifies whether to report keyboard status changes. If HFHOSTS is set, the keyboard status information is returned in the KSI private ANSI control.
	HFXLATKBD		Valid in either the KSR or the MOM mode.
hf_select[1]	Mode selectors:		
	HFWRAP		Valid only in the KSR mode
	HFLPFKS		Valid in either the KSR or the MOM mode.
	HFDIALS		Valid in either the KSR or the MOM mode.
	HFJKANA		Valid in either the KSR or the MOM mode.
	HFMOUSE		Valid in either the KSR or the MOM mode.
	HFTABLET		Valid in either the KSR or the MOM mode.
hf_value[0]	New mode values:		
	HFHOSTS	0	Does not report when the Shift key is pressed (default).
		1	Reports when Shift key is pressed.
	HFXLATKBD	0	Sends key data as untranslated key controls.
		1	Translates keyboard input (default).
hf_value[0]	New mode values:		
	HFWRAP	0	Does not wrap the cursor when the presentation space boundary is exceeded.
		1	Wraps cursor (default).
	HFLPFKS	0	Disables LPFK input (default).
		1	Enables LPFK input.
	HFDIALS	0	Disables dial (valuator) input (default).
		1	Enables dial input.
	HFJKANA	For use with Japanese licensed program only.	
		0	Disables kana shift state (default).
		1	Enables kana input.
	HFMOUSE	0	Disables the mouse from sending data (default).
		1	Enables the mouse to send data.
	HFTABLET	0	Disables the tablet from sending data (default).
		1	Enables the tablet to send data.

An attempt to set a protocol mode that is not valid results in the rejection of the entire request.

Set Keyboard LEDs

The structure for this command is the **hfkled** structure, which contains the following fields:

hf_intro.hf_typehi	HFKLEDCH.
hf_intro.hf_typelo	HFKLEDCL.
hf_sublen	2.
hf_subtype	0.
hf_ledselect	Indicates which of three LEDs to change: HFNUMLOCK Num Lock LED. HFCAPSLOCK Caps Lock LED. HFSCROLLOCK Scroll Lock LED.
hf_ledvalue	Indicates the value to which to set the LEDs specified in the hf_ledselect field. LEDs that are specified with a 1 bit are set: HFNUMLOCK Num Lock LED. HFCAPSLOCK Caps Lock LED. HFSCROLLOCK Scroll Lock LED.

Set LPFKs and Set Dial Granularities

The Set LPFKs operation turns on and off the lighted programmable function keys. The Set Dial Granularities command sets the dial granularities. *Granularity* refers to the number of events per full 360-degree revolution of the dial. The **hfdial_lpfk** structure, which contains the following fields, is used for both of these commands:

hf_intro.hf_typehi	HFLPFKSCH (Set LPFKs) or HFDIALSCH (Set Dial Granularities).
hf_intro.hf_typelo	HFLPFKSCL (Set LPFKs) or HFDIALSCL (Set Dial Granularities).
hf_sublen	2.
hf_subtype	0.
hf_mask.keys	Set LPFKs only. A 4-byte bit mask numbered from 0 through 31. Bits that are set specify the LPFK flag values to change.
hf_mask.dials	Set dial granularities only. A 4-byte bit mask numbered from 0 through 31. Bits that are set specify the dial granularity values to change.
hf_data2.lpfk.flags	Set LPFKs only. A 4-byte set of bits numbered from 0 through 31. For LPFKs selected by the hf_mask.keys field, a 0 bit disables the LPFK, and a 1 bit enables the LPFK. (LPFK 1 is the most significant bit.)
hf_data2.granularity	Set dial granularities only. An array of sixteen 1-byte values giving the granularity of each dial. The values in the array represent powers of 2, and they can range from 2 to 8.

Write Sound

This command sends output to the speaker. The **hfsound** structure is used for this command. The hf_mode byte determines whether to implement sound commands for the active virtual terminal and whether to interrupt the application after the sound command is performed. No range check is made for the frequency or duration values. The **hfsound** structure contains the following fields:

hf_intro.hf_typehi	HFSOUNDCH.
hf_intro.hf_typelo	HFSOUNDCL.

hf_sublen	2.	
hf_subtype	0.	
hf_mode	Mode:	
	HFSIGSOUND	If set, causes the **SIGSOUND** signal to be sent to the process when this sound command is run or discarded. If not set, no signal is sent
	HFEXECALWAYS	If set, causes this sound command to be run whether or not this virtual terminal is active. If not set, the sound command is run 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 from the speaker device all sound requests that belong to a process that no longer requires sound. Only the sound requests that have the HFEXECALWAYS mode off are left in the speaker device. An inactive terminal ignores this command.

Sending a Cancel Sound or Write Sound command flushes the speaker driver queue when a virtual terminal transition occurs. This does not cancel sound currently playing, but only flushes future requests. Regardless of whether the sound request is run or purged, the virtual terminal receives a response if the HFSIGSOUND mode is on. The **hfcansnd** structure, which contains the following fields, is used for this command:

hf_intro.hf_typehi	HFCANSNDCH.
hf_intro.hf_typelo	HFCANSNDCL.

Change Physical Display

This command changes the physical display to which the virtual terminal is logically attached. The **hfchgdsp** structure, which contains the following fields, is used for this command:

hf_intro.hf_typehi	HFCHGDSPCH.
hf_intro.hf_typelo	HFCHGDSPCL.
hf_sublen	2.
hf_subtype	0.
hf_mode	If the HFNONDEF flag is set, the identifier specified in the hf_devid field is used for the physical display device. If this flag is not set, the current display device is used.
hf_devid	The physical display device identifier that is returned from the **Query Physical Device** command. If the requested ID is not in the configured list of available display devices, no action is taken. If the requested ID is available, but a suitable font for the display device is not loaded, no action is taken.

Note: If the physical terminal is changed, it might also be necessary to change the **TERM** environment variable.

Also refer to the descriptions of the **hft.h** structures for query ioctl operations, special ioctl operations, read operations, KSR write operations, and MOM write operations.

HFT KSR write Operations

Several structures in the **hft.h** header file support keyboard-send-receive (KSR) high function terminal (HFT) output operations. These device-independent KSR mode operations for character-oriented applications include:

- KSR color map specification
- Font palette redefinition
- Cursor representation redefinition.

Set KSR Color Map

This command specifies the color to associate with certain display adapters. The **hfcolorpal** structure, which is used for this command, contains the following fields:

hf_intro.hf_typehi	HFCOLORPALH.
hf_intro.hf_typelo	HFCOLORPALL.
hf_sublen	2.
hf_subtype	0.
hf_numcolor	The number of entries in the color palette.
hf_palet	The array that defines the color palette settings. This field is repeated for each of the 16 adapter-specific values in the array.

Change Fonts

The **hffont** structure, which contains the following fields, is used for the change fonts request:

hf_intro.hf_typehi	HFFONTH.
hf_intro.hf_typelo	HFFONTL.
hf_sublen	2
hf_subtype	1
hf_primary	Font ID of primary font attribute.
hf_alt1	Font ID of first alternate font attribute.
hf_alt2	Font ID of second alternate font attribute.
hf_alt3	Font ID of third alternate font attribute.
hf_alt4	Font ID of fourth alternate font attribute.
hf_alt5	Font ID of fifth alternate font attribute.
hf_alt6	Font ID of sixth alternate font attribute.
hf_alt7	Font ID of seventh alternate font attribute.

Redefine Cursor Representation

The cursor representation data format determines how the cursor is presented on the display screen. The **hfcursor** structure, which contains the following fields, is used for cursor redefinition:

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.
	HFSNGLUS	Single underscore.
	HFDBLUS	Double underscore.
	HFHALFBLOB	Lower half of illuminated character cell.
	HFMIDLINE	Double mid-character line.
	HFFULLBLOB	Full illuminated character cell.

Also refer to the descriptions of the **hft.h** structures for query ioctl operations, special ioctl operations, read operations, general write operations, and MOM write operations.

HFT MOM write Operations

Several structures in the **hft.h** header file support monitor mode (MOM) high function terminal (HFT) output for complex applications that require direct access to the display hardware for graphics and other operations. Programs that operate the display hardware in all points addressable (APA) mode should select the monitor mode of the virtual terminal. The device-dependent MOM operations include:

- Screen request
- Input ring buffer definition
- Screen release.

Screen Request

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, and then receives a **SIGGRANT** signal when the screen is granted.

The screen request uses the **hfmomscreq** structure, which contains the following fields:

hf_intro.hf_len	The length of the request. This is a value of 6 if no ring buffer is defined or a value of 12 if a ring buffer is used.
hf_intro.hf_typehi	HFMOMREQH.
hf_intro.hf_typelo	HFMOMREQL.
hf_sublen	2.
hf_subtype	0.
hf_ringlen[2]	Contains the length of the ring in bytes (the size of the **hfmomring** structure, including the raw data).
hf_ringoffset[4]	Contains the offset to the input ring buffer from the beginning of the hf_ringlen field. (Initialize to the value of &ring minus the value of &hfmomscreq.hf_ringlen.)

If you do not want to specify a ring buffer, set the length of the hf_intro field to the size of the introducer, and then read the input with the standard **read** subroutine.

Input Ring Buffer Definition

The input ring buffer structure, **hfmomring**, must be aligned on a word boundary and defined in memory after the screen request (**hfmomscreq** structure). The hf_ringoffset field is the difference between the ring buffer address and the address of the hf_ringlen field, and it must be a positive value. Usually, the **hfmomring** ring buffer structure is globally 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 the hf_ringoffset field must reflect such filler bytes.

The **hfmomring** input ring buffer structure contains the following fields:

hf_intreq — Interrupt request. This field can be set to a value of 0xFF by the application to cause the HFT subsystem to send a signal each time an input event occurs. Data entry into the input ring buffer generates a **SIGMSG** signal. When the hf_intreq field is set to a value of 0 (the default), a signal is sent to the application only when the buffer goes from being empty to non-empty. This byte is automatically reset to a value of 0 by the virtual terminal each time it stores input data into the ring buffer.

hf_ovflow — Overflow. This field determines whether the input ring buffer can accommodate additional input information. A value of 0xFF indicates an overflow; a value of 0x00 indicates normal operation.

hf_source — The offset into the input ring buffer where the HFT puts received data. Because this offset starts from the beginning of the ring, the virtual terminal offset value is 32. Application programs must not alter this field after initializing it at definition time; otherwise, continuation of software function cannot be assured.

hf_sink — The offset into the **hfmomring** structure from which the application reads data. Because this offset starts from the beginning of the input ring buffer, the value is 32. The application must modify this field as input data is removed from the ring.

hf_rdata — The raw data.

Screen Release

If a MOM virtual terminal is active, pressing a hot key (the Alt–Action key sequence) for the NEXT WINDOW function causes a **SIGRETRACT** signal to be sent to the process or group of processes specified by the HFSMON type **ioctl** subroutine. Before activating the next virtual terminal, the operating system allows the program to save the state of the display hardware (such as registers and refresh memory). Then, the program must write a screen release control to the terminal to inform the operating system that the state of the display hardware has been saved. This request uses the **hfmomscrel** structure, which contains the following fields:

hf_intro.hf_len — The length of the entire structure, including the input ring buffer, minus 3.

hf_intro.hf_typehi — HFMOMRELH.

hf_intro.hf_typelo — HFMOMRELL.

After the display hardware is released, the next virtual terminal is activated. If a virtual terminal is not released within 30 seconds of the receipt of the **SIGRETRACT** signal, all processes in that terminal group receive a **SIGKILL** signal. This prevents disabled programs from disrupting the NEXT WINDOW function.

A program can issue a **pause** subroutine if there is no work to do while the display hardware is not available. When the MOM virtual terminal is reactivated (with the Alt–Action key sequence), the program receives a **SIGGRANT** signal and can resume direct output to the display monitor. The display hardware state cannot be assumed to be the same as when the program released it.

Also refer to the descriptions of the **hft.h** structures for query ioctl operations, special ioctl operations, read operations, general write operations, and KSR write operations.

Implementation Specifics

Th **hft** device driver file and **hft.h** header file are part of AIX Base Operating System (BOS) Runtime.

Files

/dev/hft	The path to the **hft** device driver special file.
/usr/include/sys/hft.h	The path to the **hft.h** header file, which defines the interface to the **hft** device driver.

Related Information

The **termio.h** header file.

The **ioctl** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

The Special Files Overview, which presents general information about special files.

The HFT Device Driver User Interface Overview in *Communication Concepts and Procedures*.

The discussion of virtual terminals in *General Concepts and Procedures*.

hia0 Special File

Purpose

Provides access to IBM Host Interface Adapter (HIA) by way of the IBM HIA device handler.

Description

The **/dev/hia0** character special file provides access to the IBM HIA device handler for the purpose of emulating 3270 display stations. The device handler is a multiplexed device handler that supports an independent logical 3270 session on each of its channels.

The device handler currently supports the 3270 mode of operation. In 3270 mode, the adapter can appear as multiple terminal sessions and is an intelligent device to the control unit. In this mode, the device handler provides the capability of emulating several IBM 3278/79 display stations.

The device handler supports one adapter, which may have up to 16 sessions and 1 invocation of the **panel20** command. The **<sys/io3270.h>** file contains the definitions of the structures used by the device handler.

Usage Considerations

When accessing the HIA device handler, the following should be taken into account:

- Driver Initialization and Termination

  The device handler may be loaded and unloaded. The device handler supports the configuration calls to initialize and terminate itself.

- Special File Support

  The subroutines other than the **open** and **close** subroutines are discussed based upon the mode in which the device handler is operating.

Subroutine Support

The HIA device handler provides HIA-specific support for the following subroutines:

- **open**
- **close**
- **read**
- **readx**
- **write**
- **writex**
- **ioctl**

The open and close Subroutines

The device handler supports the **/dev/hia0** special file as a character multiplex special file. The special file must be opened for both reading and writing (O_RDWR). There are no special considerations for closing the special file. To start the device handler for the next available port, an **open** subroutine call is made to the **/dev/hia0** file.

The read Subroutine

Data received by the communication adapter from the host is placed in the buffer until either the message completes or the buffer is full. When either condition occurs, the AIX driver returns program control back to the application. To determine the status of a **read** call, the application program can issue the WDC_INQ ioctl operation. The following are examples of **read** call status types:

- Command chaining
- More data for this particular message.

If the status returned by the WDC_INQ operation indicates more data is available, another **read** call should be issued immediately. The application program must read available data as soon as possible so as not to degrade link performance.

If a **read** call is made and no data is available, the calling process is blocked until data becomes available. To avoid blocking, the program can use the **poll** subroutine.

The host sends data as an outbound 3270 data stream.

Note: The 3270 **write** commands require the application to send status to the host indicating if the 3270 data stream is valid. Status may be sent using the WDC_SSTAT ioctl operation.

The readx Subroutine

Data received by the communication adapter from the host is placed in the buffer until either the message completes or the buffer is full. Upon completion of the **read** subroutine call, the **io3270** structure pointed to by the *ext* parameter contains the status. The status is set in the **io_flags** field of the **io3270** structure and takes the following values:

WDI_DAVAIL There is additional data for this link address.

WDI_COMM There is a communication error. The **io_status** field contains the corresponding message code.

WDI_PROG There is a program error. The **io_status** field contains the corresponding message code.

WDI_MACH There is a hardware error. The **io_status** field contains the corresponding message code.

When set, the **WDI_DAVIL** flag indicates that the data just read completes an outbound 3270 data stream. If the **WDI_DAVAIL** flag indicates there is more data available, the application should immediately issue another **readx** call. Available data must be read as soon as possible so as not to degrade link performance.

If a **readx** call is made and no data is available, the calling process is blocked until data becomes available. To avoid blocking, the **poll** subroutine may be used. The host sends data as an outbound 3270 data stream.

Note: The 3270 **write** commands require the application to send a status to the host. To send a status, the application can use the WDC_SSTAT ioctl operation.

The write Subroutine

The **write** call sends an inbound 3270 data stream to the host. The buffer specified by a **write** call must contain a complete inbound 3270 data stream. A write operation is complete when the data has been successfully transferred from the specified buffer.

The writex Subroutine

The **writex** call sends to the host an inbound 3270 data stream. The buffer specified by a **writex** call must contain a complete inbound 3270 data stream.

The **write** is complete when the data has been successfully transferred from the buffer specified on the subroutine call. Upon completion of the write, the **io3270** structure pointed to by the write extension contains the status. The status code is set in the **io_flags** field of the **io3270** structure and takes the following values:

WDI_DAVAIL There is data available for this link address. The data must be read before any write can occur.

WDI_COMM There is a communication error. The **io_status** filed contains the corresponding message code.

WDI_PROG There is a program error. The **io_status** field contains the corresponding message code.

WDI_MACH There is a hardware error. The **io_status** field contains the corresponding message code.

When data is available for reading, it should be read immediately so as to not impact performance. A **write** or **writex** subroutine call cannot be done until the data is read.

ioctl Subroutine

The following **ioctl** operations may be issued to the device handler in DFT mode.

IOCINFO Returns the logical terminal number, the EBCDIC representation of the controller type, and the controller attachment protocol in the **iocinfo** structure.

WDC_INQ Inquires the status of the last read or write call issued by the application. Also, WDC_INQ is used to determine if data is available for reading. The returned status is placed into the **io3270** structure.

If the **WDI_DAVAIL** flag is set in the **io_flags** field, one of the following status values is returned by the WDC_INQ operation.

STAT_ACK Indicates that the previously received 3270 data stream is valid and the proper response is to be made to the host.

STAT_RESET Indicates a RESET KEY should be sent.

STAT_BERR	Indicates a buffer error. An invalid buffer order or address was received.
STAT_BADC	Indicates an invalid 3270 command was received.
STAT_UNSUP	Indicates that an unsupported 3270 command was received.

Error Conditions

The following error conditions may be returned when accessing the device handler through the **/dev/hia0** special file:

EBUSY	Indicates that an open was requested for a channel that is already open.
EFAULT	Indicates that the caller specified an invalid buffer.
EINTR	Indicates that a subroutine call was interrupted.
EINVAL	Indicates that the caller specified an invalid argument.
EIO	Indicates that an unrecoverable I/O error occurred on the requested data transfer.
ENODEV	Indicates that an open was requested for an invalid channel.
ENOMEM	Indicates that the device handler could not allocate memory for use in the data transfer.
ENXIO	Indicates that an operation was requested for an invalid minor device number.

Implementation Specifics

The **hia0** special file requires the HIA device handler.

Files

/usr/include/sys/io3270.h	Defines structures used by the HIA device handler.
/usr/include/sys/devinfo.h	Contains defines and structures for devices.

Related Information

The **open** subroutine, **close** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine, **poll** subroutine.

The **panel20** command.

The Special Files Overview, which provides general information about special files.

Understanding I/O Access Through Special Files and Understanding Raw I/O Access to Block Special Files in *Kernel Extensions and Device Support Programming Concepts.*

lp Special File

Purpose

Provides access to the line printer device driver.

Description

The **lp** driver provides an interface to the port used by a printer.

Printer Modes

The **lp** driver interprets carriage returns, backspaces, line feeds, tabs, and form feeds in accordance with the modes that are set in the driver (through the **splp** command or configuration). The number of lines per page, columns per line, and the indentation at the beginning of each line can also be selected. The default for these modes can be found using the **lsattr** command. The following modes can be set with the LPRMODS **ioctl** operation:

PLOT Determines if the data stream is interpreted by the device driver when formatting the text. If the PLOT mode is off, the text is formatted using the current values set with the LPRSET **ioctl** operation.

If the PLOT mode is set, no interpretation of the data stream is performed and the bytes are sent to the printer without modification. Setting the PLOT mode causes other formatting modes, such as NOFF and NOFL, to be ignored. The default printer backend, **piobe**, sends all output in PLOT mode.

When in PLOT mode, it is the application's responsibility to send a final form feed character. If the last write operation was performed while not in PLOT mode, the final form-feed character will be sent by the device driver.

NOFF If this mode is on, a form-feed character is replaced with line-feed characters, based on the current line value set with the LPRSET **ioctl** operation. This mode is ignored if the PLOT mode is active.

NONL If this mode is on, each line-feed character is replaced with a carriage return. This mode is ignored if the PLOT mode is active.

NOCL If this mode is off, a carriage return is inserted after each line-feed. If the mode is on, no carriage return is inserted after the line feed character. This mode is ignored if the PLOT mode is active.

NOTAB If this mode off, 8 position tabs are simulated using spaces. If the NOTAB mode is on, the tab character is replaced with a space. This mode is ignored if the PLOT mode is active.

NOBS If this mode off, backspaces are sent to the printers. If the NOBS mode is on, the backspace is simulated by sending a carriage return followed by spaces to the proper print position. This mode is ignored if the PLOT mode is active.

NOCR If this mode on, each carriage return is replaced with a line feed character. This mode is ignored if the PLOT mode is active.

CAPS If this mode on, lower-case characters are converted to upper-case. This mode is ignored if the PLOT mode is active.

WRAP If this mode off, the line is truncated at the right margin and any characters received past the right margin are discarded. If the WRAP mode is on, the characters received after the right margin are printed on the next line preceded by three dots (...). This mode is ignored if the PLOT mode is active.

FONTINIT The FONTINIT mode is initially off. It is turned on by an application when a printer font has been initialized. It can turned off in the following two cases:

- An application wants fonts to be reinitialized.
- A FATAL printer error occurs. In this case, the **lp** device driver turns the FONTINIT mode off.

RPTERR If the RPTERR mode is off and an error occurs, the device driver does not return until the error has been cleared or a cancel signal is received. If the RPTERR mode is on, the device driver waits the amount of time specified by a previous LPRSTOV **ioctl** operation and then returns with an error.

Error Handling When the RPTERR Mode Is Off

If the RPTERR mode is off, no error reporting is performed. The device driver waits for the error to be cleared or a cancel signal to be received before returning to the application. This is the default mode and is intended for existing applications that do not perform error recovery.

If a signal is received by the device driver, the current operation is returned incomplete with an EINTR return code.

If printing is canceled and the printer is in PLOT mode, it is the application's responsibility to send the final form feed to eject the partial page. If the printer is not in PLOT mode, the final form feed after cancelation will be sent by the device driver.

Error Handling When the RPTERR Mode Is On

If the RPTERR mode is on, the device driver will wait for the time specified in the **v_timeout** configuration parameter and then return the uncompleted operation with an error return code. This return allows the application to get the printer status and display an error message if desired.

Note: When a device driver returns an incomplete operation with an error code (as described above), it is the application's responsibility to re-send any data not printed.

Usage Considerations

Device Dependent Subroutines

Most printer operations are implemented using the **open**, **read**, **write**, and **close** subroutines. However, these subroutines provide little or no information to the calling program about the configuration and state of the printer. The **ioctl** subroutine provides a more device-specific interface to the printer device driver.

Most of these subroutines pass data contained in structures. In all cases, a structure of the type indicated should be allocated in the calling routine. A pointer to this structure should then be passed to the device driver.

The open and close Subroutines

If an adapter for a printer is not installed, an attempt to open fails. If the printer adapter is busy, the **open** subroutine returns an error. However, all children created by a process that successfully opens the **lp** special file inherit the open printer.

The driver allows multiple **open** subroutines to occur if they all have a *mode* parameter value of read-only. Thus, the **splp** command can perform inquiries when the printer adapter is currently in use. The **lp** driver allows only one process to write to a printer adapter at a time.

The **close** subroutine waits until all output completes before returning to the user.

The read and write Subroutines

The **read** subroutine is not implemented for the native I/O parallel port.

The ioctl Subroutine

The possible **ioctl** operations and their descriptions are:

IOCINFO Returns a structure defined in the **<sys/devinfo.h>** header file, which describes the device.

LPRGET Returns the page length, width and indentation values. These values are used by the device driver when PLOT mode is not set. Note that the default printer backend, **piobe**, sends all print jobs with PLOT mode set. This **ioctl** operation uses the **lprio** structure, as defined in the **<sys/lpio.h>** header file.

LPRGETA Gets the RS232 parameters. These are the values for baud rate, character rate, character size, stop bits and parity. Refer to the **LPR232** structure and to the **termio** structure, as defined in the **termios.h** header file.

LPRMODG Gets the printer modes. These printer modes support the various formatting options and error reporting. This **ioctl** operation uses the **LPRMOD** structure, as defined in the **<sys/lpio.h>** header file.

LPRSET Sets the page length, width and indent values. These values are used by the device driver when PLOT mode is not set. Note that the default printer backend, piobe, sends all print jobs with PLOT mode set. This **ioctl** operation uses the **lprio** structure, as defined in the **<sys/lpio.h>** header file.

LPRSETA Sets the RS232 parameters. These are the values for baud rate, character rate, character size, stop bits and parity. Refer to the **LPR232** structure and to the **termio** structure, as defined in the **termios.h** header file.

LPRMODS Sets the printer modes. These printer modes support the various formatting options and error reporting. This **ioctl** operation uses the **LPRMOD** structure, as defined in the **<sys/lpio.h>** header file.

LPRGTOV Gets the current time-out value and stores it in the **lptimer** structure defined in the **<sys/lpio.h>** header file. The time-out value is measured in seconds.

LPRSTOV Sets the time-out value. The *arg* parameter to this **ioctl** operation points to a **lptimer** structure defined in **<sys/lpio.h>**. The time-out value must be given in seconds.

LPQUERY Provides access to the printer status. Refer to the **<sys/lpio.h>** header file for value definitions. The types of errors are the following:

- The printer is out of paper.
- No select bit: the printer may be turned off or not installed.
- The printer is busy.
- The printer is unknown.

Implementation Specifics

The Printer Addition Management Subsystem: Programming Overview provides more information on implementation specifics.

The **lp** special fle is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/lp

sys/lpio.h

Related Information

The **splp** command, **lsattr** command, **piobe** command.

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **ioctl** subroutine.

The Special Files Overview, which presents general information about special files.

The Printer Addition Management Subsystem: Programming Overview, Understanding I/O Access Through Special Files, Understanding Device Driver Classes, Understanding Character I/O Device Drivers, and Multiplexed Support in Character I/O Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

lvdd Special File

Purpose

Provides access to the logical volume device driver.

Description

The logical volume device driver provides character (raw) access to logical volumes. The Logical Volume Manager associates a major number with each volume group and a minor number for each logical volume in a volume group.

When performing character I/O, each request must start on a logical block boundary of the logical volume. The logical block size is 512 bytes. This means that for character I/O to a logical volume device, the offset supplied to the **lseek** subroutine must specify a multiple of 512 bytes. In addition, the number of bytes to be read or written, supplied to the **read** or **write** subroutine, must be a multiple of 512 bytes.

Block I/O requests cannot be larger than a logical track group (128K bytes) and must not cross a logical track group boundary.

Note: I/O requests should not be sent to the block special file interface when the logical volume is mounted. When a logical volume is mounted (that is, the block special file is opened by the file system), any I/O requests from the user made to that logical volume should be made only through the character special file.

Usage Considerations

Warning: POTENTIAL FOR DATA CORRUPTION OR SYSTEM CRASHES: Data corruption, loss of data, or loss of system integrity will occur if devices supporting paging, logical volumes, or mounted file systems are accessed using block special files. Block special files are provided for logical volumes and disk devices on AIX and are solely for system use in managing file systems, paging devices and logical volumes. They should not generally be used for other purposes. Additional information concerning the use of special files may be obtained in Understanding I/O access Through Special Files.

The open and close Subroutines

No special considerations.

The read and write Subroutines

Extension word specification for the readx and writex subroutines

The *ext* parameter for the **readx** and **writex** extended I/O subroutines is used to indicate specific physical or logical operations or both. The upper 4 bits of the *ext* parameter are reserved for internal LVDD use. The value of this word is defined by logically ORing values from the following list, as defined in the **<sys/lvdd.h>** header file:

WRITEV Perform physical write verification on this request. This operation can only be used with the **writex** subroutines.

RORELOC For this request, perform relocation on existing relocated defects only. Newly detected defects are not to be relocated.

MWC_RCV_OP Mirror write consistency recovery operation. This option is used by the recovery software to make consistent all mirrors with writes outstanding at the time of the crash.

NOMWC Inhibit mirror write consistency recovery for this request only. This operation can only be used with the **writex** subroutine.

AVOID_C1, AVOID_C2, AVOID_C3
For this request, avoid the specified copy (mirror). This operation can only be used with the **readx** subroutine.

RESYNC_OP For this request, synchronize the specified logical track group (LTG). This operation can only be used with the **readx** subroutine and must be the only operation.

There are some restrictions when using this operation. To synchronize a whole logical partition (LP), a series of **readx** subroutines using the RESYNC_OP operation must be done. The series must start with the first logical track group (LTG) in the partition and proceed sequentially to the last LTG. Any deviation from this will result in an error. The length provided to each **readx** operation must be exactly 128K bytes (LTG size).

Normal I/O can be done concurrently anywhere in the logical partition while the RESYNC_OP is in progress. If an error is returned, the series must be restarted from the first LTG. An error is returned only if resynchronization fails for every stale physical partition copy of any logical partition. Therefore, stale physical partitions are still possible at the end of synchronizing a LP.

Normal I/O operations do not need to supply the *ext* parameter and can use the **read** and **write** subroutines.

The ioctl Operations

The IOCINFO ioctl operation

The IOCINFO **ioctl** operation returns the **devinfo** structure, as defined in the **<sys/devinfo.h>** header file. The values returned in this structure are defined as follows for requests to the logical volume device driver:

devtype Equal to DD_DISK (as defined in the **devinfo.h** header file).

flags Equal to DF_RAND.

devsubtype Equal to DS_LV.

bytpsec Bytes per block for the logical volume.

secptrk Number of blocks per logical track group.

trkpcyl Number of logical track groups per partition.

numblks Number of logical blocks in the logical volume.

The XLATE ioctl operation

The XLATE **ioctl** operation translates a logical address (logical block number and mirror number) to a physical address (physical device and physical block number on that device). The caller supplies the logical block number and mirror number in the **xlate_arg** structure, as defined in the **<sys/lvdd.h>** header file. This structure contains the following fields:

lbn Logical block number to translate.

mirror The number of the copy for which to return a **pbn** (physical block number on disk). Possible values are:

1 copy 1 (primary)

2 copy 2 (secondary)

3 copy 3 (tertiary)

p_devt Physical **dev_t** (major/minor number of the disk)

pbn Physical block number on disk.

Error Conditions

In addition to the possible general errors returned by the **ioctl** subroutine, the following errors can also be returned from specific **ioctl** operation types.

ENXIO The logical volume does not exist. (This error type is relevant to both the IOCINFO and XLATE **ioctl** operations.)

ENXIO The logical block number is larger than the logical volume size. (This error type is relevant to the XLATE **ioctl** operation.)

ENXIO The copy (mirror) number is less than 1 or greater than the number of actual copies. (This error type is relevant to the XLATE **ioctl** operation.)

ENXIO No physical partition has been allocated to this copy (mirror). (This error type is relevant to the XLATE **ioctl** operation.)

Implementation Specifics

The Logical Volume Manager (LVM) Subsystem provides more details on implementation specifics.

The **lvdd** special file is part of AIX Base Operating System (BOS) Runtime.

Files

Logical volume special file names can be assigned by the administrator of the system. However, **/dev/lv1**, **/dev/lv2** and **/dev/rlv1**, **/dev/rlv2** are the names conventionally chosen.

Related Information

The Logical Volume Manager (LVM) Subsystem.

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **lseek** subroutine, **ioctl** subroutine.

The Special Files Overview, which presents general information about special files.

Logical Volume Storage Overview in *General Programming Concepts*.

Understanding I/O Access Through Special Files, Understanding Major and Minor Numbers For A Special File, Understanding I/O Access Through Special Files, Understanding Device Driver Classes, Understanding Block I/O Device Drivers, and Understanding Character I/O Device Drivers, and Providing Raw I/O Support in *Kernel Extensions and Device Support Programming Concepts*.

mem Special File and kmem Special File

Purpose

Provides privileged virtual memory read and write access.

Description

The **/dev/mem** and **/dev/kmem** character special files provide access to a pseudo-device driver that allows read and write access to system memory or I/O address space. Typically, these special files are used by operating system utilities and commands (such as **crash**, **sar**, **iostat**, and **vmstat**) to obtain status and statistical information about the system.

Programs accessing these special files must have appropriate privilege. Commercial application programs should avoid using the **mem** and **kmem** files, since the virtual memory image is quite specific to the operating system level and machine platform. Use of these special files thus seriously affects the portability of the application program to other operating systems and machine platforms.

Warning: When incorrect access to virtual memory is made through these files, process termination, a system crash, or loss of system data integrity can result.

Usage Considerations

kmem Special File Access

The **kmem** special file provides access to the virtual memory address space for the current process, as it is seen by the kernel. The seek offset, set by the **lseek** subroutine, is used to specify the virtual address targeted for the read or write. The **kmem** pseudo device driver only supports the **open**, **close**, **read**, **readx**, **writex**, and **write** subroutines.

The **knlist** system subroutine is typically used to obtain the addresses of kernel symbols to read or write through access provided by the **kmem** special file.

Before issuing a read or write operation, the **lseek** subroutine must be used to designate the relevant starting address in virtual memory. If this address is within the first two gigabytes of address space, then the **read** or **write** subroutine calls can be used. However, if the upper two gigabytes of address space are to be accessed, the **readx** and **writex** form of the subroutine calls must be used. In this case, the *ext* (extension) parameter must be set to a value of TRUE. This causes the lseek offset to be interpreted relative to the upper 2 gigabytes of address space.

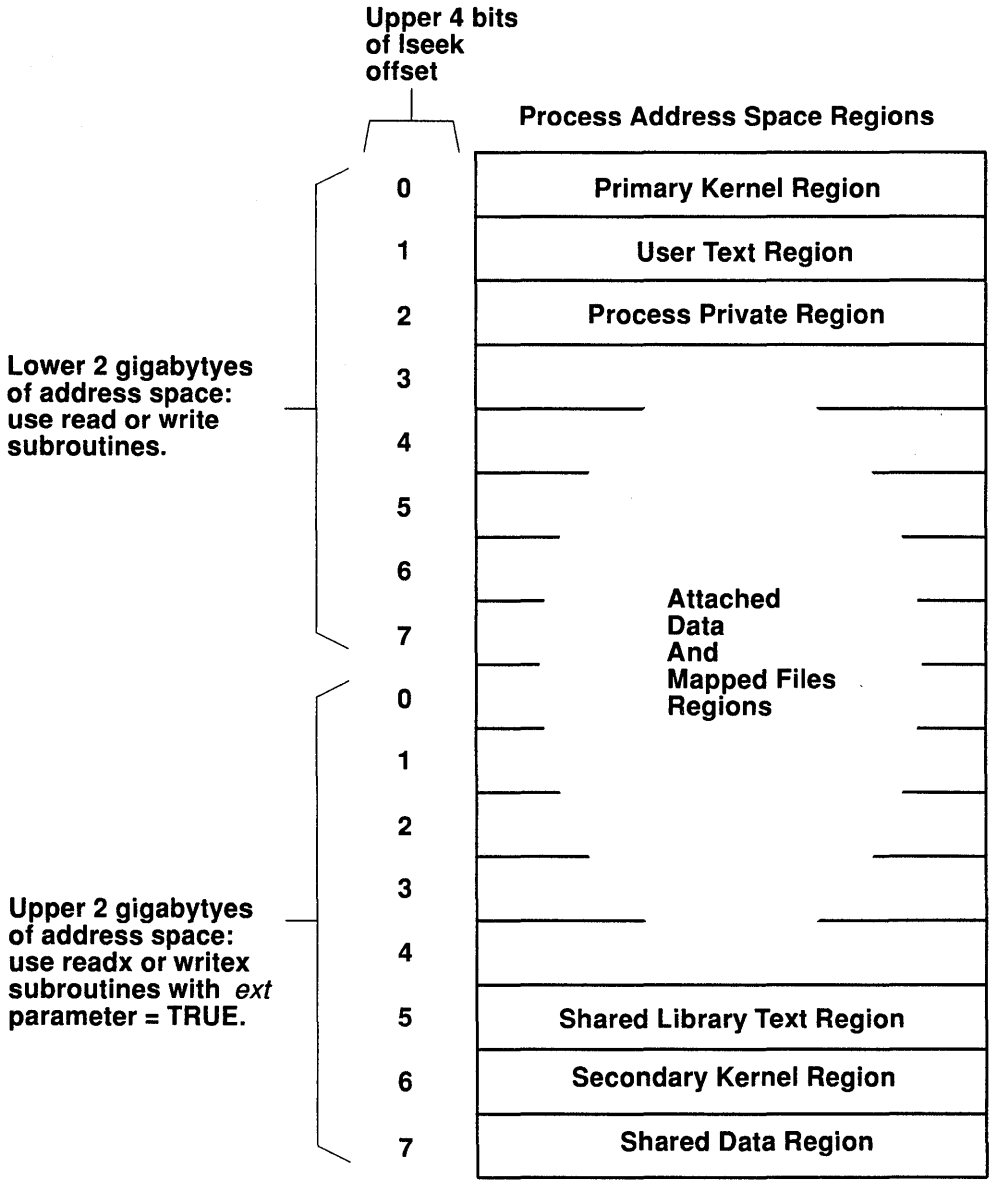
On the RISC System/6000 machine platform, the process address space is defined as shown in the Implementation Specifics section. This address space layout can vary on other machine platforms and versions of the operating system.

mem Special File Access

The **mem** special file access is specific to the machine platform on which the operating system is executing. Use of this special file by application programs should be strictly avoided, as it is provided for diagnostic and problem determination procedures only. Please refer to the Implementation Specifics section for details on the function provided by this special file.

Process Address Space Regions for the kmem Special File

The Process Address Space Map illustrates the layout of process address space regions as accessed through the **kmem** special file on the RISC System 6000 system:



Implementation of mem Special File Access

The **mem** special file has traditionally provided direct access to physical memory. This capability and its interface requirements are machine-specific. However, for AIX on the RISC System/6000, this function is indirectly provided by using the *ext* (extension) parameter on the **readx** and **writex** subroutine calls. When issuing a **readx** or **writex** subroutine call associated with the **/dev/mem** special file, the *ext* parameter must contain a valid segment register value as defined in the *POWERstation and POWERserver Hardware Technical Reference — General Information*. This allows the program to access all physical memory mapped by the page table as well as the system I/O address space.

The seek offset set by the **lseek** subroutine call is used to specify the address offset within the segment described by the *ext* parameter. The upper four bits of the offset are not used. The pseudo device driver only supports the **open**, **close**, **read**, **readx**, **write**, and **writex** subroutine calls. The **lseek** subroutine call must also be used before issuing the **readx** or **writex** subroutine calls in order to specify the address offset.

If the **read** or **write** subroutine calls are used with this special file, the access to memory is identical to that provided by the **kmem** special file.

Implementation Specifics

The **kmem** and **mem** special files are part of AIX Base Operating System (BOS) Runtime.

Files

/dev/mem

/dev/kmem

Related Information

The **crash** command, **sar** command, **iostat** command, **vmstat** command.

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **lseek** subroutine, **ioctl** subroutine, **select** subroutine, **poll** subroutine, **knlist** subroutine.

Memory and I/O address space addressing is described in the *POWERstation and POWERserver Hardware Technical Reference — General Information*.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files, Understanding Character I/O Device Drivers, Multiplexed Support in Character I/O Device Drivers, and Understanding Pseudo-Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

mpqn Special File

Purpose

Provides access to multiprotocol adapters by way of the Multiprotocol (MPQP) device handler.

Description

The **/dev/mpq***n* character special file provides access to the MPQP device handler for the purpose of providing access to a synchronous network. The MPQP device handler supports multiple adapters.

Usage Considerations

When accessing the MPQP device handler, the following should be taken into account:

- Driver Initialization and Termination

  The device handler may be loaded and unloaded. The handler supports the configuration calls to initialize and terminate itself.

- Special File Support

  Calls other than the **open** and **close** subroutine calls are discussed based on the mode in which the device handler is operating.

Subroutine Support

The MPQP device handler supports the **open**, **close**, **read**, **write**, and **ioctl** subroutines in the following manner:

- The **open** and **close** subroutines

  The device handler supports the **/dev/mpq***n* special file as a character–multiplex special file. The special file must be opened for both reading and writing (O_RDWR). There are no particular considerations for closing the special file. The special file name used in an **open** call differs depending upon how the device is to be opened. Types of special file names are:

  /dev/mpq*n* Starts the device handler for the selected port.

  /dev/mpq*n***/D** Starts the device handler for the selected port in Diagnostic mode.

- The **read** subroutine

  Can take the form of a **read, readx**, **readv**, or **readvx** subroutine call. For this call, the device handler copies the data into the buffer specified by the caller.

- The **write** subroutine

  Can take the form of a **write, writex**, **writev**, or **writevx** subroutine call. For this call, the device handler copies the user data into a buffer and transmits the data on the LAN.

- The **ioctl** subroutine

  CIO_START Starts a session and registers a network ID.

  CIO_HALT Halts a session and removes a network ID.

  CIO_QUERY Returns the current RAS counter values. These values are defined in the **<sys/comio.h>** and **<sys/ecluser.h>** files.

CIO_GET_STAT	Returns the current adapter and device handler status.
MP_START_AR	Puts the MPQP port into Auto–response mode.
MP_STOP_AR	Permits the MPQP port to exit Auto–response mode.
MP_CHG_PARMS	Permits the data link control (DLC) to change certain profile parameters after the MPQP device has been started.
MP_SET_DELAY	Sets the value of NDELAY.

Error Conditions

The following error conditions may be returned when accessing the device handler through the **/dev/mpq***n* special file:

ECHRNG Indicates that the channel number is out of range.

EAGAIN Indicates that the maximum number of DMAs was reached allowed and the handler cannot get memory for internal control structures.

EBUSY Indicates one of the following:

- The port is not in correct state.
- The port should be configured, but it is not opened or started.
- The port state is not opened for start of an ioctl operation.
- The port is not started or in data transfer state.

EIO Indicates that the handler could not queue command to adapter.

EFAULT Indicates that the cross memory copy service failed.

EINTR Indicates that the sleep was interrupted by a signal.

EINVAL Indicates one of the following:

- The port not set up properly.
- The handler could not set up structures for write.
- The port is invalid.
- A select operation was called by a kernel process.
- The specified physical link parameter is invalid for that port.
- The read was called by a kernel process.

ENOMEM Indicates one of the following:

- No mbuf or mbuf clusters are available.
- The total data length is more than a page.
- There is no memory for internal structures.

ENOMSG Indicates that the status queue pointer is null, and there are no entries.

ENOTREADY Indicates that the port state in define device structure (DDS) is not in Data Transfer mode or that the implicit halt of port failed.

ENXIO Indicates one of the following:

- The port has not been started successfully.
- An invalid adapter number was specified.
- The channel number is illegal.
- The adapter is already open in Diagnostic mode.
- The ACB pointer is null or does not exist.
- The registration of the interrupt handler failed.
- The port does not exist or is not in proper state.
- The adapter number is out of range.

The communication device handler chapter defines specific errors returned on each subroutine call.

Implementation Specifics

This file functions with the Multiprotocol device handler.

File

/usr/include/sys/devino.h Contains device types and information.

Related Information

The **open** subroutine, **close** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access to Special Files, Understanding Major and Minor Numbers for a Special File, and Understanding Raw I/O Access to Block Special Files in *Kernel Extensions and Device Support Programming Concepts*.

null Special File

Purpose

Provides access to the null device, typically for writing to the bit bucket.

Description

The **null** special file provides character access to the null device driver. This device driver is normally accessed to write data to the bit bucket (when the data is to be discarded).

Usage Considerations

The open and close Subroutines

The null device can be opened by using the **open** subroutine with the **/dev/null** special file name. The **close** subroutine should be used when access to the null device is no longer required.

The read and write Subroutines

Data written to this file is discarded. Reading from this file always returns 0 bytes.

The ioctl Subroutine

There are no **ioctl** operations available for use with the **null** special file. Any **ioctl** operation issued returns with the **ENODEV** error type.

Implementation Specifics

The **null** special file is part of AIX Base Operating System (BOS) Runtime.

File

/dev/null

Related Information

The **open** subroutine, **close** subroutine, **ioctl** subroutine.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files, Understanding Character I/O Device Drivers, and Understanding Pseudo-Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

nvram Special File

Purpose

Provides access to platform-specific nonvolatile RAM used for system boot, configuration, and fatal error information. This access is achieved through the machine I/O device driver.

Description

The **nvram** character special file provides access to the machine pseudo-device driver for accessing or modifying machine-specific nonvolatile RAM. The appropriate privilege is required to open this special file. This special file is used by machine-specific configuration programs to store or retrieve configuration and boot information using the nonvolatile RAM or ROM provided on the machine.

The **/dev/nvram** character special file allows character mode access to the nonvolatile RAM and selected parts of ROM as a multiplexed device where each channel provides access to a separately allocated section of nonvolatile memory. When opening the **nvram** special file as a multiplexed device, the special file name is **/dev/nvram/n**, where *n* is the channel number (for example, **/dev/nvram/0**). This special file supports **open**, **close**, **read**, **write**, and **ioctl** operations.

The machine device driver also allows read access of the compressed PCAT BIOS code provided in system nonvolatile memory. To read the PCAT BIOS code, this special file must be opened with the channel name of **pcbios** as follows:

```
/dev/nvram/pcbios
```

For additional information concerning the use of this special file to access machine specific nonvolatile RAM, please refer to the Machine Device Driver information.

Note: Application programs should not generally access other channels supported by this special fle. Because nonvolatile RAM is platform-specific, any reliance on its presence and implementation places portability constraints upon the using application. In addition, accessing the nonvolatile RAM may cause loss of system boot and configuration information. Such loss could require system administrative or maintenance task work to rebuild or recover.

Usage Considerations

The open and close Subroutines

The machine device driver supports the **/dev/nvram** special file as a multiplexed character special file. This special file name in conjunction with the **pcbios** channel name can be used to access the machine device driver to read the compressed PCAT BIOS code stored in the system read-only memory (ROM).

This compressed PCAT BIOS code may be uncompressed and used by an application program to assist in providing PC simulation on the RISC System/6000 platform. Multiple concurrent opens to this channel are not supported by the machine device driver and return an error.

The read, write, and lseek Subroutines

The **read** subroutine is supported after a successful open of the **/dev/nvram** special file with a channel name of **pcbios**. The **read** operation starts transferring data at the location in nonvolatile memory associated with the PCAT BIOS code with an offset specified by the offset value associated with the file pointer being used on the subroutine.

On a **read** subroutine, if the end of the data area is reached before the transfer count is reached, the number of bytes read before the end of the data area was reached is returned. If the read starts after the end of the data area, an error of ENXIO is returned by the driver.

The **lseek** subroutine may be used to change the starting read offset within the non-volatile memory data area associated with the compressed PCAT BIOS code. The **write** subroutine is not supported on this channel and results in an error return of ENODEV.

The ioctl Subroutine

The following **ioctl** commands can be issued to the machine device driver after a successful open of the **pcbios** channel using the /**dev/nvram** special file:

IOCINFO Returns machine device driver information in the caller's **devinfo** structure, as pointed to by the *arg* parameter to the **ioctl** subroutine. This structure is defined in the **sys/devinfo.h** header file. The device type for this device driver is DD_PSEU.

Error Conditions

The following error conditions can be returned when accessing the machine device driver using the /**dev/nvram** special file name:

EBUSY An open operation was requested for a channel that is already open.

EFAULT A buffer specified by the caller was invalid on a **read**, **write** or **ioctl** subroutine call.

ENXIO A read operation was attempted past the end of the data area specified by the channel.

ENODEV A write operation was attempted to the read-only **pcbios** channel.

ENOMEM A request was made with a user-supplied buffer that is too small for the requested data.

Implementation Specifics

For additional information concerning the data areas accessed by other channels associated with the /**dev/nvram** special file, refer to the Machine Device Driver documentation in the Device Configuration Subsystem: Programming Introduction.

The **nvram** special fle is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/nvram/0, /dev/nvram/1, ... /dev/nvram/n

Related Information

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **lseek** subroutine, **ioctl** subroutine.

The format of the compressed PCAT BIOS code provided in system nonvolatile memory is described in the *POWERstation and POWERserver Hardware Technical Reference — General Information.*

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special FilesUnderstanding Device Driver Classes, Understanding Block I/O Device Drivers, Providing Raw I/O Access in a Block Device Driver, Providing Raw I/O Support, Understanding Character I/O Device Drivers, Multiplexed Support in Character I/O Device Drivers, Understanding Pseudo-Device Drivers, and The Device Configuration Subsystem: Programming Introduction in *Kernel Extensions and Device Support Programming Concepts.*

pty Special File

Purpose

Provides the pseudo terminal (PTY) device driver.

Description

The **pty** device driver provides support for a *pseudo terminal.* A pseudo terminal includes a *control* and *slave* pair of character devices. The slave device provides to processes essentially the same interface as provided by the **tty** device driver. However, instead of providing support for a hardware device, the slave device is manipulated by another process through the control half of the pseudo terminal. That is, anything written on the control device is given to the slave device as input and anything written on the slave device is presented as input on the control device.

In the AIX Version 2 and Berkeley PTY subsystems, commands had to search for an unused pseudo terminal by opening each control side sequentially. The control side could not be opened if it was already in use. Thus, the opens would fail, setting the **errno** variable to EIO, until an unused pseudo terminal was found. The current version of AIX simplifies this process by using two multiplexed special files, **/dev/ptc** and **/dev/pts**. A *multiplexed special file* refers to a special file with an added attribute that indicates a channel number. When the multiplexed file is opened directly, the **pty** device driver allocates an unused channel and opens the associated file. A multiplexed file can also be opened by entering an extension after the normal path name. The **pty** device driver interprets the extension as a request to open the specified channel. (For example, if `/dev/ptc/23` is entered, the **pty** device driver opens **ptc** channel 23.)

By opening the multiplexed files directly with **/dev/ptc**, an application can quickly open the control and slave sides of an unused pseudo terminal. The name of the corresponding slave side can be retrieved using the **ttyname** subroutine, which always returns the name of the slave side, even if the control side was opened first.

When the **pty** device driver is configured into the system, a system management parameter for the pty device driver determines the number of symbolic links that are made to the multiplexed special files. For example, `/dev/ptyp0` is linked to `/dev/ptc/0` and `/dev/ttyp0` is linked to `/dev/pts/0`. These symbolic links are created for applications that use the Berkeley PTY naming convention when addressing pseudo terminals.

The following **ioctl** commands apply to pseudo terminals:

TIOCSTOP — Stops output to a terminal. This is the same as typing the Ctrl–s key combination. No parameters are allowed for this command.

TIOCSTART — Restarts output that was stopped by a TIOCSTOP command or by the Ctrl–s key combination. No parameters are allowed for this command.

TIOCPKT — Enables and disables the packet mode. Packet mode is enabled by specifying (by reference) a non-zero parameter. It is disabled by specifying (by reference) a zero parameter. When applied to the control side of a pseudo terminal, each subsequent read from the terminal returns data written on the slave part of the pseudo terminal preceded either by a zero byte (symbolically defined as TIOCPKT_DATA) or by a single byte that reflects control status information. In the latter case, the byte is an inclusive or of zero or more of the following bits:

TIOCPKT_FLUSHREAD	The read queue for the terminal is flushed.
TIOCPKT_FLUSHWRITE	The write queue for the terminal is flushed.
TIOCPKT_STOP	Output to the terminal is stopped with Ctrl–s.
TIOCPKT_START	Output to the terminal is restarted.
TIOCPKT_DOSTOP	The stop character as defined by the current TTY line discipline is Ctrl–s; the start character as defined by the line discipline is Ctrl–q.
TIOCPKT_NOSTOP	The start and stop characters are not Crtl–s and Ctrl–q.

While this mode is in use, the presence of control status information to be read from the control side can be detected by a select for exceptional conditions.

This mode is used by the **rlogin** command and the **rlogind** command to log in to a remote host and implement remote echoing and local Ctrl–s and Ctrl–q flow control with proper back-flushing of output.

TIOCUCNTL Enables and disables a mode that allows a small number of simple user **ioctl** commands to be passed through the pseudo terminal, using a protocol similar to that of the TIOCPKT mode. The TIOCUCNTL and TIOCPKT modes are mutually exclusive.

This mode is enabled from the control side of a pseudo terminal by specifying (by reference) a non-zero parameter. It is disabled by specifying (by reference) a zero parameter. Each subsequent read from the control side will return data written on the slave part of the pseudo terminal preceded either by a zero byte or by a single byte that reflects a user control operation on the slave side. A user control command consists of a special **ioctl** operation with no data; the command is given as UIOCCMD(*Value*), where the *Value* parameter is a number in the range 1 through 255. The operation value is received as a single byte on the next read from the control side. A value of 0 (zero) can be used with the UIOCCMD **ioctl** to probe for the existence of this facility; the zero is not made available for reading by the master side. Command operations can be detected with a select for exceptional conditions.

TIOCREMOTE A mode for the control half of a pseudo terminal, independent of TIOCPKT. This mode causes flow control rather than input editing to be implemented for input to the pseudo terminal, regardless of the terminal mode. Each write to the control terminal produces a record boundary for the process reading the terminal. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an end-of-file character. This mode is used for remote line editing in a window manager and flow controlled input.

Implementation Specifics

This device driver is provided for Berkeley compatibility.

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/pty[p–r][0–9a–f]	The control pseudo terminal.
/dev/tty[p–r][0–9a–f]	The slave pseudo terminal.

Related Information

The **ioctl** subroutine, **ttyname** subroutine.

The **rlogin** command, **rlogind** command.

For general information about special files, refer to the Special Files Overview. For general information about the TTY subsystem, refer to the TTY Subsystem Overview in *General Programming Concepts*.

rhdisk Special File

Purpose

Provides raw I/O access to the physical volumes (fixed-disk) device driver.

Description

The **rhdisk** special file provides raw I/O access and control functions to the physical disk device drivers for the physical disks on the RISC System/6000 machine platforms. Raw I/O access is provided through the **/dev/rhdisk0**, **/dev/rhdisk1**, ..., character special files.

Direct access to physical disks through block special files should be avoided. Such access can impair performance and also cause data consistency problems between data in the block I/O buffer cache and data in system pages. The **/dev/hdisk** block special files are only provided for system use in managing file systems, paging devices and logical volumes.

The prefix of **r** on the special file name indicates that the drive is to be accessed as a raw device rather than a block device. Performing raw I/O with a fixed-disk requires that all data transfers be in multiples of the disk block size. Also, all **lseek** subroutines that are made to the raw disk device driver must result in a file pointer value that is a multiple of the disk block size.

Usage Considerations

Warning: POTENTIAL FOR DATA CORRUPTION OR SYSTEM CRASHES: Data corruption, loss of data, or loss of system integrity will occur if devices supporting paging, logical volumes, or mounted file systems are accessed using block special files. Block special files are provided for logical volumes and disk devices on AIX and are solely for system use in managing file systems, paging devices and logical volumes. They should not generally be used for other purposes. Additional information concerning the use of special files may be obtained in Understanding I/O access Through Special Files.

The open and close Subroutines

The **openx** subroutine is supported to provide additional functions to the open sequence. The **openx** operation requires appropriate permission to execute. Attempting to execute this subroutine without the proper permission results in a return value of –1 with the **errno** global variable set to the value EPERM.

The read and write Subroutines

The **readx** and **writex** subroutines are supported to provide for additional parameters affecting the raw data transfer. The *ext* parameter is used to specify certain options that apply to the request being made. The options are constructed by logically ORing zero or more of the following values:

WRITEV Perform physical write verification on this request. This operation can only be used with the **writex** subroutine.

HWRELOC Perform hardware relocation of the specifed block before the block is written. This is done only if the drive supports safe relocation. Safe relocation ensures that once the relocation is started, it will complete safely regardless of power outages. This operation can only be used with the **writex** subroutine.

UNSAFEREL Perform hardware relocation of the specifed block before the block is written. This is done if the drive supports any kind of relocation (safe or unsafe). This operation can only be used with the **writex** subroutine.

The ioctl Subroutine

One **ioctl** operation is defined for all device drivers that support the **ioctl** subroutine call. This is the IOCINFO operation. The remaining **ioctl** operations are all specific to physical disk devices. Diagnostic mode is not required for the IOCINFO operation.

The IOCINFO ioctl operation

The IOCINFO **ioctl** operation returns a structure (defined in the **<sys/devinfo.h>** header file) for a device type of DD_DISK.

Error Conditions

In addition to those errors listed for the **ioctl**, **open**, **read**, and **write** subroutines, the following other error types are possible:

EACCES An **open** subroutine call has been made to a device in diagnostic mode.

EACCES A diagnostic **openx** subroutine call has been made to a device already opened.

EACCES A diagnostic **ioctl** operation has been attempted when not in diagnostic mode.

EINVAL An *nbyte* parameter to a **read** or **write** subroutine is not a multiple of the disk block size.

EINVAL An unsupported **ioctl** operation has been attempted.

EINVAL An unsupported **readx** or **writex** operation has been attempted.

EMEDIA The target device has indicated an unrecovered media error.

ENXIO A parameter to the **ioctl** subroutine is invalid.

ENXIO A **read** or **write** subroutine has been attempted beyond the end of the disk.

EIO The target device cannot be located or is not responding.

EIO The target device has indicated an unrecovered hardware error.

EMFILE An **open** operation has been attempted for an adapter that already has the maximum permissible number of opened devices.

EPERM The caller lacks the appropriate privilege.

Implementation Specifics

The SCSI Subsystem Programming Introduction and SCSI **scdisk** device driver provide further information on implementation specifics.

The **rhdisk** special file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/rhdisk0, /dev/rhdisk1, ... /dev/rhdiskn
/dev/hdisk0, /dev/hdisk1, ... /dev/hdiskn

Related Information

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **lseek** subroutine, **ioctl** subroutine.

The SCSI **scdisk** device driver.

The Special Files Overview, which presents general information about special files.

The SCSI Subsystem Programming Introduction in *General Programming Concepts.*

Understanding I/O Access Through Special Files, Understanding Device Driver Classes, Understanding Block I/O Device Drivers, Understanding Character I/O Device Drivers, and Providing Raw I/O Support in *Kernel Extensions and Device Support Programming Concepts.*

rmt Special File

Purpose

Provides access to the sequential access bulk storage medium device driver.

Description

Magnetic tapes are used primarily for backup, file archives, and other off-line storage. Tapes are accessed through the **/dev/rmt0**, ... , **/dev/rmt255** special files. The **r** in the special file name indicates *raw* access through the character special file interface. A tape device does not lend itself well to the category of a block device. Thus, only character interface special files are provided.

Special files associated with each tape device determine what action is taken during open or close operations. These files also dictate, for applicable devices, at what density data is to be written to tape. The following figure shows the names of these special files and their corresponding characteristics.

Special File Name	Rewind–on–Close	Retension–on–Open	Bytes per Inch
/dev/rmt*	Yes	No	HIGH
/dev/rmt*.1	No	No	HIGH
/dev/rmt*.2	Yes	Yes	HIGH
/dev/rmt*.3	No	Yes	HIGH
/dev/rmt*.4	Yes	No	LOW
/dev/rmt*.5	No	No	LOW
/dev/rmt*.6	Yes	Yes	LOW
/dev/rmt*.7	No	Yes	LOW

Note: The density value (bytes per inch) is ignored when using a magnetic tape device that does not support multiple densities. For tape drives that do support multiple densities, the density value only applies when writing to the tape. When reading, the drive defaults to the density at which the tape is written.

For the 9-track tape drive, the special file with the low-density attribute specifies the tape is to be written at 1600 bpi. The special file with the high-density attribute specifies the tape is to be written at 6250 bpi.

For the quarter-inch (QIC) tape drive, the special file with the low-density attribute specifies the tape is to be written using the QIC-120 standard density. The special file with the high-density attribute specifies the tape is to be written using the QIC-150 standard density.

Usage Considerations

Most tape operations are implemented using the **open**, **read**, **write**, and **close** subroutines. However, for diagnostic purposes, use of the **openx** is required.

The open and close Subroutines

Care should be taken when closing a file after writing. If the application reverses over the data just written, no file marks will be written. However, for tape devices that allow for block update, unless the application spaces in the reverse direction or returns the tape position to the beginning of the tape (BOT), one or two file marks will be written upon closing the device. (The number of file marks depends on the special file type.)

For multi-tape jobs, the special file must be opened and closed for each tape. The user is not allowed to continue if the special file is opened and the tape has been changed.

The **openx** operation is intended primarily for use by the diagnostic commands and utilities. Appropriate authority is required for execution. Executing this subroutine without the proper authority results in an **openx** return value of –1 with the **errno** global variable set to the **EPERM** value.

The read and write Subroutines

When opened for reading or writing, the tape is assumed to be positioned as desired. When the tape is opened as no-rewind-on-close (**/dev/rmt*.1**) and a file is written, a single file mark is written upon closing the tape. When the tape is opened as rewind-on-close (**/dev/rmt***) and a file is written, a double tape file mark is written upon closing the tape. When the tape is opened as no-rewind-on-close and reads from a file, the tape is positioned upon closing after the end of file (EOF) mark following the data just read.

By specifically choosing the **rmt** file, it is possible to create multiple file tapes.

Although tapes are accessed through character interface special files, the number of bytes required by either a read or write operation must be a multiple of the blocksize defined for the magnetic tape device.

During a read, the record size is returned as the number of bytes read, up to the buffer size specified. If an EOF condition is encountered, then a zero-length read is returned, with the tape positioned after the EOF.

An end of media (EOM) condition encountered during a read or write operation results in the return of the number of bytes successfully ready or written. When a write is attempted after the device has reached EOM, a value of –1 is returned with the **errno** global variable set to the ENXIO value. When a read is attempted after the device has reached EOM, a zero-length read is returned. Successive reads continue to return a zero-length read.

Data Buffering With a Tape Device

Some tape devices contain a data buffer to maximize date transfer speed when writing to tape. It is useful to note that a write operation sent to tape is returned as complete when the data has been transferred to the tape device data buffer. The data in the buffer is then written to tape asynchronously. This permits increased data transfer speed since the host need not wait for I/O completion.

Two modes are provided by the tape device driver to facilitate use of these data buffers. The non-buffered mode causes writes to tape to bypass the data buffer and go directly to tape. In buffered mode, all write subroutines are returned as complete when the transfer data has been successfully written to the tape device buffer. The device driver does not flush the data buffer until the special file is closed or EOM is encountered.

If an EOM is encountered while running in buffered mode, the device attempts to flush the device data buffer. It is possible for the residual count to exceed the write transfer length in the case of buffered mode. In some cases, the flushing of residual data may actually run the

tape off the reel. Either case is considered a failure and results in a return value of –1 with the **errno** global variable set to the value EIO. These errors can require the user to run in non-buffered mode.

Note: The following points must be taken into consideration when using the **rmt** special file:

1. Failures that result in a device reset while reading or writing to tape require the special file to be closed and the job restarted. Any commands issued after this condition occurs and until the special file is closed will result in a return value of –1 and **errno** is set to the value EIO. Non-reset type errors (that is, media or hardware errors) result in the tape being left positioned where the error occurred.
2. For multi-tape jobs, the special file must be opened and closed for each tape. The user is not allowed to continue if the special file is opened and the tape has been changed.
3. A signal received by the tape device driver will cause the current command to abort. This allows the application to halt time-consuming commands (that is, an erase operation) without re-cycling the drive power or waiting for a time-out to occur.

The ioctl Subroutine

One **ioctl** operation is defined for all device drivers that use the **ioctl** subroutine. This is the IOCINFO operation. For the **rmt** special file, the STIOCTOP operation has also been defined.

The IOCINFO ioctl operation

The IOCINFO **ioctl** operation returns a structure defined in the **<sys/devinfo.h>** header file.

The STIOCTOP ioctl operation

The STIOCTOP **ioctl** operation provides for command execution such as erase and retension. The parameter to the **ioctl** subroutine using the STIOCTOP operation is the address of a **stop** structure, as defined in the **<sys/tape.h>** header file.

The operation found in the **st_op** field in the stop structure is performed **st_count** times, except with rewind, erase, and retension. The available operations are:

STREW	Rewind.
STERASE	Erase tape; leave at load point.
STRETEN	Retension tape; leave at load point.
STWEOF	Write and end-of-file mark.
STFSF	Forward space file.
STFSR	Forward space record.
STRSF	Reverse space file.
STRSR	Reverse space record.

Note: Execution of the preceding commands is dependent on the particular tape device and which commands are supported. If the command is not supported on a particular device, a value of –1 is returned with the **errno** global variable set to the value EINVAL.

Error Conditions

In addition to general error types listed for **ioctl**, **open**, **read**, and **write** subroutines, the following specific error types may also occur:

EAGAIN	An open operation was attempted to a device that is already open.
EBUSY	The target device is reserved by another initiator.
EINVAL	O_APPEND is supplied as a mode in which to open.
EINVAL	An *nbyte* parameter to a **read** or **write** subroutine is not an even multiple of the blocksize.
EINVAL	A parameter to the **ioctl** subroutine is invalid.
EINVAL	The requested **ioctl** operation is not supported on the current device.
EIO	Could not space forward or reverse **st_count** records before encountering EOM or a file mark.
EIO	Could not space forward or reverse **st_count** file marks before encountering EOM.
EMEDIA	The tape device has encountered an unrecoverable media error.
ENXIO	A write operation was attempted while the tape is at end of media.
ENOTREADY	There is no tape in the drive or the drive is not ready.
EPERM	The requested subroutine requires appropriate authority.
ETIMEDOUT	A command has timed out.
EWRPROTECT	An open operation for read/write was attempted on a read-only tape.
EWRPROTECT	An **ioctl** operation that effects media was attempted on a read-only tape.

Implementation Specifics

The SCSI **rmt** device driver provides further information about implementation specifics.

The **rmt** special file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/rmt0, /dev/rmt0.1, /dev/rmt0.2, ..., /dev/rmt0.7
/dev/rmt1, /dev/rmt1.1, /dev/rmt1.2, ..., /dev/rmt1.7 ...
/dev/rmt255, /dev/rmt255.1, /dev/rmt255.2, ..., /dev/rmt255.7

Related Information

The **open** subroutine, **close** subroutine, **read** subroutine, **write** subroutine, **ioctl** subroutine.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files , Understanding Device Driver Classes, Understanding Block I/O Device Drivers, and Understanding Character I/O Device Drivers in *Kernel Extensions and Device Support Programming Concepts.*

scsi Special File

Purpose

Provides access to the SCSI adapter driver.

Description

The **scsi** special file provides an interface to an attached SCSI adapter. This special file should not be opened directly by application programs (with the exception of diagnostics applications).

Usage Considerations

The SCSI Adapter Device Driver provides information on using the SCSI adapter.

Implementation Specifics

The SCSI Adapter Device Driver provides information on implementation specifics of the SCSI adapter.

The **scsi** special file is part of AIX Base Operating System (BOS) Runtime.

File

/dev/scsi0, **/dev/scsi1**, ...

Related Information

The Special Files Overview, which presents general information about special files.

Understanding I/O Access Through Special Files and SCSI Subsystem Programming Introduction in *Kernel Extensions and Device Support Programming Concepts*.

tokn Special File

Purpose

Provides access to the Token-Ring adapters by way of the Token-Ring device handler.

Description

The **tok***n* character special file provides access to the Token-Ring device handler which provides access to a Token-Ring Local Area Network. The device handler supports up to four Token-Ring adapters.

Usage Considerations

When accessing the Token-Ring device handler, the following should be taken into account:

- Driver Initialization and Termination

  The device handler may be loaded and unloaded. The device handler supports the configuration calls to initialize and terminate itself.

- Special File Support

  Calls other than the **open** and **close** subroutines are discussed based on the mode in which the device handler is operating.

Subroutine Support

The Token-Ring device handler provides specific support for the **open**, **close**, **read**, **write**, and **ioctl** subroutines.

The open and close Subroutines

The device handler supports the **/dev/tok***n* special file as a character-multiplex special file. The special file must be opened for both reading and writing (O_RDWR). There are no particular considerations for closing the special file. The special file name used in an **open** call differs depending upon how the device is to be opened. Types of special file names are:

/dev/tok*n* Starts the device handler for the selected port, where the value of *n* is 0 <= *n* <= 7.

/dev/tok*n***/D** Starts the device handler for the selected port in Diagnostic mode, where the value of *n* is 0 <= *n* <= 7.

/dev/tok*n***/W** Starts the device handler for the selected port in Diagnostic Wrap mode, where the value of *n* is 0 <= *n* <= 7.

The read Subroutine

Can take the form of a **read, readx**, **readv**, or **readvx** subroutine. For this call, the device handler copies the data into the buffer specified by the caller.

The write Subroutine

Can take the form of a **write**, **writex**, **writev**, or **writevx** subroutine. For this call, the device handler copies the user data into a kernel buffer and transmits the data on the LAN.

The ioctl Subroutine

The Token-Ring device handler supports the following ioctl operations:

IOCINFO Returns a structure of device information to the user specified area. The **devtype** field is DD_NET_DH and the **devsubtype** field is DD_TR, as defined in the **<sys/devinfo.h>** file.

CIO_START Starts a session and registers a network ID.

CIO_HALT Halts a session and removes a network ID from the network ID table.

CIO_QUERY Returns the current counter values, as defined in the **<sys/comio.h>** and **<sys/tokuser.h>** file.

CIO_GET_STAT Returns current adapter and device handler status.

TOK_QVPD Returns adapter vital product data.

TOK_GRP_ADDR Allows the setting of the active group address for the Token-Ring adapter.

TOK_FUNC_ADDR Allows the setting of a functional address for the Token-Ring adapter.

TOK_RING_INFO Returns information about the Token-Ring device.

Error Conditions

The following error conditions may be returned when accessing the device handler through the **dev/tok***n* special file:

EACCES Indicates that permission to access the adapter is denied for one of the following reasons:

- Device has not been configured.
- Diagnostic mode open request denied.
- The call is from a kernel-mode process.

EAGAIN Indicates that the transmit queue is full.

EBUSY Indicates one of the following:

- The device is already opened in Diagnostic mode.
- The maximum number of opens has already been reached.
- The request is denied.
- The device is in use.
- The device handler cannot terminate.

EEXIST Indicates that the device is already configured or the device handler is unable to remove the device from switch table.

EFAULT Indicates that the an invalid address or parameter was specified.

EINTR	Indicates that the subroutine was interrupted.
EINVAL	Indicates one of the following: • The parameters specified were invalid. • The define device structure (DDS) is invalid. • The device handler is not in Diagnostic mode.
ENOCONNECT	Indicates that the device has not been started.
ENETDOWN	Indicates that the network is down and the device handler is unable to process the command.
ENOENT	Indicates that there was no DDS available.
ENOMEM	Indicates that the device handler was unable to allocate required memory.
ENOMSG	Indicates that there was no message of desired type.
ENOSPC	Indicates that the network ID table is full or the maximum number of opens was exceeded.
EADDRINUSE	Indicates that the specified network ID is in use.
ENXIO	Indicates that the specified minor number was invalid.
ENETUNREACH	Indicates that the device handler is in Network Recovery mode and is unable to process the write operation.
EMSGSIZE	Indicates that the data is too large for the supplied buffer.

Implementation Specifics

This file functions with the Token-Ring Device Handler.

Related Information

The **open** subroutine, **close** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine.

Token-Ring Device Handler Overview in *Communications Programming Concepts.*

POWERstation and POWERserver Hardware Technical Reference — Options and Devices.

The Special Files Overview, which presents general information about special files.

Understanding I/O Access to Special Files, Understanding Major and Minor Numbers For A Special File, and Understanding Raw I/O Access to Block Special Files in *Kernel Extensions and Device Support Programming Concepts*

trace Special File

Purpose

Supports event tracing.

Syntax

#include <sys/trcctl.h>

Description

The **/dev/systrace** and **/dev/systrcctl** special files support the monitoring and recording of selected system events. Minor device 0 of the **trace** drivers is the interface between processes that record trace events and the trace daemon. Trace events are written to the **/dev/systrace** file by the **trchk** and **trcgen** subroutines and the **trcgenk** kernel service. Minor devices 1 through 7 of the **trace** drivers support generic trace channels for tracing system activities such as communications link activities.

Implementation Specifics

The **trace** special file is part of AIX Base Operating System (BOS) Runtime.

Files

/dev/systrace

/dev/systrcctl

Related Information

The **trcgenk** kernel service.

The Special Files Overview, which presents general information about special files.

RAS Kernel Services, Understanding I/O Access Through Special Files, Understanding Major and Minor Numbers, Understanding Character I/O Device Drivers, Multiplexed Support in Character I/O Device Drivers, and Understanding Pseudo-Device Drivers in *Kernel Extensions and Device Support Programming Concepts*.

tty Special File

Purpose

Supports the controlling terminal interface.

Description

For each process, the **tty** special file is a synonym for the controlling terminal associated with that process. By directing messages to the **tty** file, application programs and shell sequences can ensure that the messages are written to the terminal even if output is redirected. Programs can also direct their display output to this file so that it is not necessary to identify the active terminal.

A terminal can belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal can be the controlling terminal for one session at most. If a session leader has no controlling terminal and opens a terminal device file that is not already associated with a session (without using the O_NOCTTY option of the **open** subroutine), the terminal becomes the controlling terminal of the session leader. If a process that is not a session leader opens a terminal file or the O_NOCTTY option is used, that terminal does not become the controlling terminal of the calling process. When a controlling terminal becomes associated with a session, its foreground process group is set to the process group of the session leader.

The controlling terminal is inherited by a child process during a **fork** function. A process cannot end the association with its controlling terminal by closing all of its file descriptors associated with the controlling terminal if other processes continue to have the terminal file open. A process that is not already the session leader or a group leader can break its association with its controlling terminal using the **setsid** function; other processes remaining in the old session retain their association with the controlling terminal.

When the last file descriptor associated with a controlling terminal is closed (including file descriptors held by processes that are not in the controlling terminal's session), the controlling terminal is disassociated from its current session. The disassociated controlling terminal can then be acquired by a new session leader.

A process can also remove the association it has with its controlling terminal by opening the **tty** file and issuing the following **ioctl** command:

```
ioctl (FileDescriptor, TIOCNOTTY, 0):
```

It is often useful to disassociate server processes from their controlling terminal so that they cannot receive input from or be stopped by the terminal.

Implementation Specifics

This device driver supports the POSIX and Berkeley line disciplines as well as the AIX Version 2 compatibility mode.

This file is part of AIX Base Operating System (BOS) Runtime.

File

/dev/tty The path to the **tty** file.

Related Information

The **fork** subroutine, **open** subroutine, **setsid** subroutine.

Chapter 4. Header Files

Header Files Overview

Information that is needed by several different files or functions as well as information that is likely to change is collected into a *header file*. A header file contains C-language definitions and structures. Centralizing this type of information into a header file facilitates the creation and update of programs. Because **#include** statements are used to insert header files into a C-language program, header files are often referred to as *include files*.

Header files can provide the following functions:

- Defining the structures of certain files and subroutines
- Defining type definition (typedef) synonyms for data types
- Defining system parameters or implementation characteristics
- Defining constants and macros that are substituted during the C language preprocessing phase.

By convention, the names of header files end with **.h** (dot h). The **.h** suffix is used by header files that are provided with the AIX operating system; however, the suffix is not required for user-generated header files.

Note: A few of the header files provided with the AIX operating system end with **.inc** (include file).

Several header files are provided with the AIX operating system. Most of these can be found in either the **/usr/include** directory or the **/usr/include/sys** directory. The AIX **pg** command can be used to view the contents of a header file. More information about the following header files is provided in this documentation:

a.out.h	Refer to the **a.out** file format.
acct.h	Refer to the **acct** file format.
ar.h	Refer to the **ar** file format.
audit.h	Refer to the **audit** file format.
core.h	Refer to the **core** file format.
dirent.h	Describes the format of a file system-independent directory entry.
fcntl.h	Defines values for the **fcntl** and **open** subroutines.
flock.h	Defines the file control options.
fullstat.h	Describes the data structure returned by the **fullstat** and **ffullstat** subroutines.
hft.h	Defines the interface to the HFT device driver.
limits.h	Defines implementation limits identified by the IEEE POSIX 1003 standard.
math.h	Defines math subroutines and constants
mode.h	Defines the interpretation of a file mode.
param.h	Defines certain hardware-dependent parameters.
poll.h	Defines the **pollfd** structure used by the **poll** subroutine.
sem.h	Describes the structures that are used by subroutines that perform semaphore operations.
sgtty.h	Defines structures used by the Berkeley terminal interface.
SNA luxsna.h	Defines structures used by the Berkeley terminal interface.

srcobj.h Defines structures used by the System Resource Controller (SRC) subsystem.

stat.h Describes the data structure returned by the status subroutines.

statfs.h Describes the structure of the statistics returned by the status subroutines.

termio.h Defines structures used by the terminal interface for the AIX Version 2 compatibility mode.

termios.h Defines structures used by the POSIX terminal interface.

types.h Defines primitive system data types.

unistd.h Defines POSIX implementation characteristics.

utmp.h Defines the format of certain user and accounting information files.

values.h Defines hardware-dependent values.

vmount.h Describes the structure of a mounted file system.

x25sdefs.h Contains the structures used by the X25 application programming interface.

3270 Host Connection Program/6000 (HCON) Header Files

HCON fxconst.inc
Defines HCON **fxter** function constants for Pascal language file transfers.

HCON fxfer.h Defines HCON **fxc** and **fxs** data structures for C language file transfers.

HCON fxfer.inc Contains HCON **fxc** and **fxs** records for Pascal language file transfers.

HCON fxhfile.inc
Contains HCON external declarations for Pascal language file transfers.

HCON g32_api.h
Contains HCON API symbol definitions and data structures for the C language.

HCON g32const.inc
Defines HCON API constants for the Pascal language.

HCON 32hfile.inc
Contains HCON API external definitions for the Pascal language.

HCON g32_keys.h
Enables HCON API in Mode_3270 for C language subroutines.

HCON g32keys.inc File
Contains common HCON API key value definitions for the Pascal language.

HCON g32types.inc File
Defines HCON API data types for the Pascal language.

Socket Header Files

in.h Defines Internet constants and structures.

nameser.h Contains Internet nameserver information.

netdb.h Contains data definitions for socket subroutines.

resolv.h Contains resolver global definitions and variables.

socket.h Contains data definitions and socket structures.

socketvar.h Defines the kernel structure per socket and contains buffer queues.

un.h Defines structures for the Interprocess Communication domain.

Example

If a C program contains the following statement, the text of the **mode.h** file is inserted into the program when it is compiled.

```
#include <sys/mode.h>
```

In the C preprocessing stage, the angle brackets (< >) are interpreted as **/usr/include/**, so that this statement includes the **mode.h** file from the **/usr/include/sys** directory.

Related Information

The **pg** command.

The File Formats Overview, which defines and describes file formats in general and lists file formats discussed in this documentation.

The Special Files Overview, which defines and describes special files in general and lists special files discussed in this documentation.

dirent.h File

Purpose

Describes the format of a file system-independent directory entry.

Description

The **dirent.h** header file describes the format of a directory entry without reference to the type of underlying file system.

The **dirent** structure is used in the directory access operations. Using these access operations and the **dirent** structure along with its associated constants and macros shields you from the details of implementing a directory and provides a consistent interface to directories across all types of file systems.

The **dirent** structure defined in the **dirent.h** header file contains the following members for each directory entry:

```
ulong_t         d_offset;       /* actual offset of this entry */
ino_t           d_ino;          /* inode number of entry */
ushort_t        d_reclen;       /* length of this entry */
ushort_t        d_namlen;       /* length of string in d_name
char d_name[_D_NAME_MAX+1];     /* name of entry (filename) */
```

_D_NAME_MAX is a constant that indicates the maximum number of bytes in a file name for all file systems. (Related to this constant is PATH_MAX, which is the maximum number of bytes in the full path name of a file, also not including the terminating null byte.)

The value of _D_NAME_MAX is specific to each file system type. It can only be determined via the **pathconf** and **fpathconf** subroutines.

The size of a **dirent** structure depends on the number of bytes in the file name.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

_D_NAME_MAX and PATH_MAX are maximum file names and path names defined across all types of file systems. Each file system can define constants applicable only to that specific file system. However, using file system-specific constants and directory structures makes it very difficult to port code across different types of file systems.

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/dirent.h The path to the **dirent.h** header file.

Related Information

The **pathconf, fpathconf** subroutines.

The **dir** file.

fcntl.h File

Purpose

Defines file control options.

Description

The **fcntl.h** header file defines the values that can be specified for the *Command* and *Argument* parameters of the **fcntl** subroutine, and for the *Oflag* parameter of the **open** subroutine. The file status flags of an open file are described in the following paragraphs.

The following flag values are accessible only to the **open** subroutine:

O_RDONLY Read only.

O_WRONLY Write only.

O_RDWR Read and write.

O_CREAT Open with file create (uses the third **open** argument).

O_TRUNC Open with truncation.

O_EXCL Exclusive open.

O_NOCTTY Do not assign a controlling terminal.

O_RSHARE Read shared open.

O_NSHARE Read shared open.

The following mask is used to determine the file access mode:

O_ACCMODE

The following file status flags are accessible to both the **open** subroutine and the **fcntl** subroutine:

O_NONBLOCK POSIX non-blocking I/O.

FNONBLOCK POSIX non-blocking I/O.

O_APPEND An append with writes guaranteed at the end.

FAPPEND An append with writes guaranteed at the end.

O_SYNC Synchronous write option.

FSYNC Synchronous write option.

FASYNC Asynchronous I/O.

O_NDELAY Non-blocking I/O.

FNDELAY Non-blocking I/O.

The following file status flags are accessible to the **open** subroutine:

O_DEFER Defered update.

O_DELAY Open with delay.

The following file descriptor flag is accessible to the **fcntl** subroutine:

FD_CLOEXEC Close this file during an exec.

The *Command* values for the **fcntl** subroutine (the **fcntl** subroutine requests) are:

F_DUPFD Duplicate the file description.

F_GETFD Get the file description flags.

F_SETFD Set the file description flags.

F_GETFL Get the file flags.

F_SETFL Set the file flags.

F_GETLK Get the file lock.

F_SETLK Set the file lock.

F_SETLKW Set the file lock and wait.

F_GETOWN Get the descriptor owner.

F_SETOWN Set the descriptor owner.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

File

/usr/include/fcntl.h The path to the **fcntl.h** header file.

Related Information

The **fcntl** subroutine, **open** subroutine.

flock.h File

Purpose

Defines file control options.

Description

The **flock** structure in the **flock.h** header file, which describes a lock, contains the following fields:

l_type — Describes the type of lock. If the value of the *Command* parameter to the **fcntl** subroutine is F_SETLCK or F_SETLCKW, the l_type field indicates the type of lock to be created. Possible values are:

F_RDLCK	A read lock is requested.
F_WRLCK	A write lock is requested.
F_UNLCK	Unlock. An existing lock is to be removed.

If the value of the *Command* parameter to the **fcntl** subroutine is F_GETLCK, the l_type field describes an existing lock. Possible values are:

F_RDLCK	A conflicting read lock exists.
F_WRLCK	A conflicting write lock exists.
F_UNLCK	No conflicting lock exists.

l_whence — Defines the starting offset. The value of this field indicates the point from which the relative offset, the l_start field, is measured. Possible values are:

SEEK_SET	The relative offset is measured from the start of the file.
SEEK_CUR	The relative offset is measured from the current position.
SEEK_END	The relative offset is measured from the end of the file.

These values are defined in the **unistd.h** header file.

l_start — Defines the relative offset in bytes, measured from the starting point in the l_whence field.

l_len — Specifies the number of consecutive bytes to be locked.

l_sysid — Contains the ID of the node that already has a lock placed on the area defined by the **fcntl** subroutine. This field is returned only when the value of the *Command* parameter is F_GETLK.

l_pid — Contains the ID of a process that already has a lock placed on the area defined by the **fcntl** subroutine. This field is returned only when the value of the *Command* parameter is F_GETLK.

l_vfs — Specifies the file system type of the node identified in the l_sysid field.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/sys/flock.h The path to the **flock.h** header file.

Related Information

The **unistd.h** file.

The **fcntl** subroutine, **lockf** subroutine, **flock** subroutine, **openx** subroutine.

fullstat.h File

Purpose

Defines the data structure returned by the **fullstat** subroutine.

Description

The **fullstat.h** header file defines the data structure that is returned by the **fullstat** and **ffullstat** subroutines. This file also defines the *Command* parameters that are used by the **fullstat** and **ffullstat** subroutines. The **fullstat** structure contains the following fields:

st_dev	ID of device containing a directory entry for this file. The file serial and the device ID uniquely identify the file within the system.
st_ino	File serial number.
st_mode	The mode of the file, as defined in the **mode.h** header file mode.
st_nlink	Number of links to file.
st_uid	User ID of the owner of the file.
st_gid	Group ID of the file owner group.
st_rdev	ID of this device. This field is defined only for character or block special files.
st_size	File size in bytes.
st_atime	Time of last access.
st_mtime	Time of last data modification.
st_ctime	Time of last file status change.
st_blksize	Optimal block size for the file system.
st_blocks	Number of blocks actually allocated to the file.
st_vfstype	File system type as defined in the **vmount.h** header file.

Time is measured in seconds since 00:00:00 GMT, January 1, 1970.

fst_type	V-node type.
fst_vfs	Virtual file system ID.
fst_flag	Indicates whether directory or file is a virtual mount point.
fst_i_gen	Generation number of the i-node.
fst_reserved[8]	Reserved.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

The following fields are maintained for source level compatibility with previous versions of AIX:

fst_uid_rev_tag

fst_gid_rev_tag

fst_nid

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/include/sys/fullstat.h	The path to the **fullstat.h** header file.
/usr/include/sys/mode.h	The path to the **mode.h** header file, in which the valid file modes are defined.
usr/include/sys/stat.h	The path to the **stat.h** header file. The **fullstat.h** header file contains an **include** statement for this file.
/usr/include/sys/types.h	The path to the **types.h** header file. The **fullstat.h** header file contains an **include** statement for this file.
/usr/include/sys/vmount.h	The path to the **vmount.h** header file.
usr/include/sys/vnode.h	The path to the **vnode.h** header file.

Related Information

The **ffullstat, fullstat** subroutines.

The **mode.h** file, **stat.h** file, **statfs.h** file, **types.h** file, **vmount.h** file.

limits.h File

Purpose

Defines implementation limits identified by IEEE POSIX 1003.

Description

The **limits.h** header file contains definitions required by the ANSI X3.159–198x Programming Language C Standard and the Institute of Electrical and Electronics Engineers (IEEE) P1003.1 Portable Operating System Interface for Computer Environments (POSIX) standard.

The constants required by the ANSI C Standard describe the sizes of basic data types, as follow:

Symbol	Value	Explanation
CHAR_BIT	8	The number of bits in a variable of type char.
CHAR_MAX	255	The maximum value of a variable of type char.
CHAR_MIN	0	The minimum value of a variable of type char.
INT_MAX	2,147,483,647	The maximum value of a variable of type int.
INT_MIN	–2,147,483,648	The minimum value of a variable of type int.
LONG_MAX	2,147,483,647	The maximum value of a variable of type long.
LONG_MIN	–2,147,483,648	The maximum value of a variable of type long.
SCHAR_MAX	127	The maximum value of a variable of type signed char.
SCHAR_MIN	–128	The minimum value of a variable of type signed char.
SHRT_MAX	32,767	The maximum value of a variable of type short.
SHRT_MIN	–32,768	The maximum value of a variable of type short.
UCHAR_MAX	255	The maximum value of a variable of type unsigned char.
UINT_MAX	4,294,967,295	The maximum value of a variable of type unsigned int.
ULONG_MAX	4,294,967,295	The maximum value of a variable of type unsigned long.
USHRT_MAX	65,535	The maximum value of a variable of type unsigned short.

Run-Time Invariant Values

The first set of values required by POSIX, run-time invariant values, are simple constants determined by basic operating system data structure sizes.

Symbol	Value	Explanation
MAX_INPUT	256	No fewer than the number of bytes specified by the MAX_INPUT symbol will be allowed in a terminal input queue.
NGROUPS_MAX	32	Maximum size of the concurrent group list.

PASS_MAX	256	Maximum number of bytes in a password (not including the terminating NULL).
PID_MAX	INT_MAX	Maximum value for a process ID.
UID_MAX	ULONG_MAX	Maximum value for a user or group ID.

Run-Time Invariant Values (Possibly Indeterminate)

The second set of run-time invariant values required by POSIX specify values that might vary, especially due to system load, but can be attained on a lightly-loaded system.

Symbol	Value	Explanation
ARG_MAX	16384	Maximum length (bytes) of arguments for the **exec** subroutine, including the environment.

The argument list and environment are allowed to consume all of the user data segment.

CHILD_MAX	40	Maximum number of simultaneous processes per user ID.

This is configurable by the **setquota** subroutine, and can never be less than 6.

MAX_CANON	256	Maximum number of bytes in a canonical input line.
OPEN_MAX	2000	Maximum number of files that one process can have open at any given time.

Path Name Variable Values

The third set of values required by POSIX, path name variable values, represent constraints imposed by the file system on file path names. Further constraints on these values might be imposed by the underlying file system implementation. Use the **pathconf** or **fpathconf** subroutine to determine any file implementation characteristics specific to the underlying file system.

Symbol	Value	Explanation
NAME_MAX	undefined	Maximum number of bytes in a file component name (not including the terminating NULL).
PATH_MAX	512	Maximum number of bytes in a path name (not including a terminating NULL).

Run-time Increasable Values

The fourth set of values required by POSIX specify values that might be increased at run time. Use the **pathconf** or **fpathconf** subroutine to determine any file implementation characteristics specific to the underlying file system.

Symbol	Value	Explanation
LINK_MAX	32,767	Maximum value of a file's link count (SHRT_MAX).
PIPE_BUF	32768	Maximum number of bytes guaranteed to be written automatically to a pipe.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is provided for POSIX compatibility.

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/include/limits.h	The path to the **limits.h** header file.
/usr/include/values.h	The path to the **values.h** header file, which defines hardware-dependent values.

Related Information

The **exec** subroutine, **pathconf** or **fpathconf** subroutine.

The **values.h** file.

math.h File

Purpose

Defines math subroutines and constants.

Description

The **math.h** 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.

Among other things, the **math.h** file defines the following constant, which is used as an error-return value:

HUGE_VAL The maximum value of a single-precision floating-point number.

If you define the __MATH__ preprocessor variable before including the **math.h** file, the **math.h** file defines macros that make the names of certain math subroutines appear to the compiler as *__xxxx*. The following names are redefined to have the __ prefix:

exp	sin	asin
log	cos	acos
log10	tan	atan
sqrt	fabs	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 subroutines directly. The __MATH__ variable is defined by default.

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*.
M_LOG10E	The base-10 logarithm of *e*.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	pi, the ratio of the circumference of a circle to its diameter.
M_PI_2	The value of pi divided by 2.
M_PI_4	The value of pi divided by 4.
M_1_PI	The value of 1 divided by pi.
M_2_PI	The value of 2 divided by pi.
M_2_SQRTPI	The value of 2 divided by the positive square root of pi.
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

File

/usr/include/math.h The path to the **math.h** header file.

Related Information

The **values.h** file.

mode.h File

Purpose

Defines the interpretation of a file mode.

Description

This version of the AIX operating system supports a 32 bit mode, which is divided into 3 parts. The 16 most significant bits are reserved by the system. The least significant 16 bits define the type of file (S_IFMT) and the permission bits. The 12 permission bits can be changed via the **chmod** subroutine or **chacl** subroutine. The file type cannot be changed.

File Type Bits

The file type determines the operations that can be applied to the file (including implicit operations, such as searching a directory or following a symbolic link). The file type is established when the file is created; it cannot be changed. The following file types are supported:

S_IFDIR	Defines a directory.
S_IFREG	Defines a regular file.
S_IFIFO	Defines a pipe.
S_IFCHR	Defines a character device.
S_IFBLK	Defines a block device.
S_IFLNK	Defines a symbolic link.
S_IFSOCK	Defines a socket.

The S_IFMT format mask constant can be used to mask off a file type from the mode.

File Attribute Bits

The file attribute bits affect the interpretation of a particular file. With some restrictions, file attributes can be changed by the owner of a file or by a privileged user. The file attribute bits are:

Attribute	Description
S_ISUID Bit	
setuid	When a process runs a regular file that has the S_ISUID bit set, the effective user ID of the process is set to the owner ID of the file. The setuid attribute can be set only by a process on a trusted path. If the file or its access permissions are altered, the S_ISUID bit is cleared.
S_ISGID (S_ENFMT) Bit	
setgid	When a process runs a regular file that has both the S_ISGID bit and the S_IXGRP permission bit set, the effective user ID of the process is set to the group ID of the file. The setgid attribute can be set only by a process on a trusted path. If the owner is establishing this attribute, the group of the file must be the effective group ID or in the concurrent group set of the process. If the file or its access permissions are altered, the S_ISGID bit is cleared.

enforced locking
: If a regular file has the S_ISGID bit set and the S_IXGRP permission bit cleared, locks placed on the file with the **lockfx** subroutine are enforced locks.

S_IFMPX Bit

multiplexed
: A character device with the S_IFMPX attribute bit set is a multiplexed device. This attribute is established when the device is created.

S_ISVTX Bit

sticky
: If a directory has the S_SVTX bit set, no processes can link to the files in that directory. Only the owner of the file or the owner of the directory can remove a file in a directory that has this attribute.

S_IXACL Bit

access control list
: If any file has the S_IXACL bit set, it can have an extended Access Control List (ACL). Specifying this bit when setting the mode with the **chmod** command causes the permission bits information in the mode to be ignored. Extended ACL entries are ignored if this bit is cleared. This bit can be implicitly cleared by the **chmod** subroutine.

S_ITCB Bit

trusted
: Any file that has the S_ITCB bit set is part of the trusted computing base. Only files in the trusted computing base can acquire privilege on a trusted path, and only files in the trusted computing base are run by the trusted shell (which is invoked with the **tsh** command). This attribute can be established or cleared only by a process running on the trusted path.

S_IJRNL Bit

journalled
: Any file that has the S_IJRNL bit set is defined as a journalled file. Updates to a journalled file are added to a log atomically. All directories and system files have the journalled attribute. This attribute cannot be reset.

File Permission Bits

The file permission bits control which processes can perform operations on a file. This includes read, write, and execute bits for the file owner, the file group, and the default. These bits should not be used to set access control information; the ACL should be used instead. The file permission bits are:

S_IRWXU
: Permits the owner of a file to read, write, and execute (run) the file.

S_IRUSR
: Permits the owner of a file to read the file.

S_IREAD
: Permits the owner of a file to read the file.

S_IWUSR
: Permits the owner of a file to write to the file.

S_IWRITE
: Permits the owner of a file to write to the file.

S_IXUSR
: Permits the owner of a file to execute (run) the file or to search the file's directory.

S_IEXEC	Permits the owner of a file to execute (run) the file or to search the file's directory.
S_IRWXG	Permits a file's group to read, write, and execute (run) the file.
S_IRGRP	Permits a file's group to read the file.
S_IWGRP	Permits a file's group to write to the file.
S_IXGRP	Permits a file's group to execute (run) the file or to search the file's directory.
S_IRWXO	Permits others to read, write, and execute (run) the file.
S_IROTH	Permits others to read the file.
S_IWOTH	Permits others to write to the file.
S_IXOTH	Permits others to execute (run) the file or to search the file's directory.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/include/sys/acl.h	The path to the **acl.h** header file, which defines the format of an Access Control List.
/usr/include/sys/access.h	The path to the **access.h** header file, which defines the constants used for access mode specification.
/usr/include/sys/mode.h	The path to the **mode.h** header file.
/usr/include/sys/stat.h	The path to the **stat.h** header file, which defines the structure of the file status information returned by certain subroutines.
/usr/include/sys/types.h	The path to the **types.h** header file, which defines the primitive data types.

Related Information

The **stat.h** file, **types.h** file.

The **chmod** command, **tsh** command.

param.h File

Purpose

Describes system parameters.

Description

Certain parameters vary for different hardware that uses the AIX operating system. These parameters are defined in the **param.h** file. The most significant parameters are:

NCARGS — Indicates the maximum number of characters, including terminating null characters that can be passed using the **exec** subroutine.

UBSIZE — The unit used by the statistics subroutines for returning block sizes of files.

This file also contains macros for manipulating machine-dependent fields.

Programs that are intended to comply with the POSIX standard should include the **limits.h** file rather than the **param.h** file.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/sys/limits.h — The path to the **limits.h** header file, which defines parameters for compatiblity with the POSIX standard.

/usr/include/sys/param.h — The path to the **param.h** header file.

Related Information

The **exec** subroutine.

poll.h File

Purpose

Defines the structures and flags used by the **poll** subroutine.

Description

The **poll.h** header file defines several structures used by the **poll** subroutine. An array of **pollfd** or **pollmsg** structures or a **pollist** structure specify the file descriptors or pointers and message queues for which the **poll** subroutine checks the I/O status. This file also defines the returned events flags, error returned events flags, device-type flags and input flags used in polling operations.

When performing a polling operation on both file descriptors and message queues, the *ListPointer* parameter points to a **pollist** structure, which can specify both file descriptors or pointers and message queues. The program must define the **pollist** structure in the following form:

```
struct pollist {
        struct pollfd[f];
        struct pollmsg[m];
};
```

The **pollfd** structure and the **pollmsg** structure in the **pollist** structure perform the following functions:

pollfd[*f*] This structure defines an array of file descriptors or file pointers. The *f* parameter specifies the number of elements in the array.

pollmsg[*m*] This structure defines an array of message queue identifiers. The *m* parameter specifies the number of elements in the array.

A POLLIST macro is also defined in the **poll.h** header file to define the **pollist** structure. The format of the macro is:

```
POLLIST(f, m) Declarator . . . ;
```

The *Declarator* parameter is the name of the variable that is declared as having this type.

The **pollfd** and **pollmsg** structures defined in the **poll.h** header file contain the following fields:

fd Specifies a valid file descriptor or file pointer to the **poll** subroutine. If the value of this field is negative, this element is skipped.

msgid Specifies a valid message queue ID to the **poll** subroutine. If the value of this field is negative, this element is skipped.

events The events being tracked. This is any combination of the following flags:

POLLIN	Input is present on the file or message queue.
POLLOUT	The file or message queue is capable of accepting output.
POLLPRI	An exceptional condition is present on the file or message queue.

revents — Returned events. This field specifies the events that have occurred. This can be any combination of the events requested by the events field. This field can also contain one of the following flags:

- POLLNVAL — The value specified by the fd field or the msgid field is neither a valid file descriptor or pointer nor the identifier of an accessible message queue.
- POLLERR — An error condition arose on the specified file or message queue.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

File

/usr/include/sys/poll.h The path to the **poll.h** header file.

Related Information

The **poll** subroutine, **select** subroutine.

The **fp_poll** kernel service, **fp_select** kernel service, **selnotify** kernel service.

sem.h File

Purpose

Describes the structures that are used by subroutines that perform semaphore operations.

Description

The **sem.h** header file defines the structures that are used by the **semop** subroutine and the **semctl** subroutine to perform various semaphore operations.

The **sem** structure stores the values that the *Commands* parameter of the **semctl** subroutine gets and sets. This structure contains the following fields:

semval — Specifies the operation permission structure of a semaphore. The data type of this field is unsigned short.

sempid — Specifies the last process that performed a **semop** subroutine. The data type of this field is pid_t.

semncnt — Specifies the number of processes awaiting semval > cval. The data type of this field is unsigned short.

semzcnt — Specifies the number of processes awaiting semval = 0. The data type of this field is unsigned short.

The **sembuf** structure stores semaphore information used by the **semop** subroutine. This structure contains the following fields:

sem_num — Specifies a semaphore on which to perform some semaphore operation. The data type of this field is unsigned short.

sem_op — Specifies a semaphore operation to be performed on the semaphore specified by the sem_num field and the *Semid* parameter of the **semop** subroutine. This value can be a positive integer, a negative integer, or zero:

- *i* — If the current process has write permission, the positive integer value of this field is added to the semval of the semaphore.
- *– i* — If the current process has write permission, a negative integer value in this field causes one of the following actions:

  If semval is greater than or equal to the absolute value of the sem_op field, the absolute value of the sem_op field is subtracted from semval.

  If semval is less than the absolute value of the sem_op field and the IPC_NOWAIT flag is set, the **semop** subroutine returns a value of –1 and sets the global variable **errno** to EAGAIN.

If semval is less than the absolute value of the sem_op field and the IPC_NOWAIT flag is not set, the **semop** subroutine increments the semncnt associated with the specified semaphore and suspends execution of the calling process until one of the following conditions is met.

- The value of semval becomes greater than or equal to the absolute value of the sem_op field. When this occurs, the value of semncnt associated with the specified semaphore is decremented, the absolute value of the sem_op field is subtracted from semval and, if SEM_UNDO is set in the sem_flg field, the absolute value of the sem_op field is added to the *Semadj* value of calling process for the specified semaphore.
- The *SemaphoreID* for which the calling process is awaiting action is removed from the system (see the **semctl** subroutine). When this occurs, the global variable **errno** is set equal to EIDRM, and a value of –1 is returned.
- The calling process receives a signal that is to be caught. When this occurs, the value of semncnt associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in the **sigaction** subroutine.

0 If the current process has read permission, a value of zero in this field causes one of the following actions:

If semval is 0, the **semop** subroutine returns a value of 0.

If semval is not equal to 0 and the IPC_NOWAIT flag is set, the **semop** subroutine returns a value of –1 and sets the global variable **errno** to EAGAIN.

If semval is not equal to 0 and the IPC_NOWAIT flag is not set, the **semop** subroutine increments the semzcnt associated with the specified semaphore and suspends execution of the calling process until one of the following conditions is met.

- The value of semval becomes 0, at which time the value of semzcnt associated with the specified semaphore is decremented.
- The *SemaphoreID* for which the calling process is awaiting action is removed from the system. When this occurs, the global variable **errno** is set equal to EIDRM, and a value of –1 is returned.
- The calling process receives a signal that is to be caught. When this occurs, the value of semzcnt associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in the **sigaction** subroutine.

The data type of the sem_op field is short.

sem_flg If the value of this field is not zero for an operation, it is constructed by logically ORing one or more of the following values:

SEM_UNDO — Specifies whether to modify the *Semadj* values of the calling process.

If this value is set for an operation and the value of the sem_op field is a positive integer, the value of the sem_op field is subtracted from the *Semadj* value of the calling process.

If this value is set for an operation and the value of the sem_op field is a negative integer, the absolute value of the sem_op field is added to the *Semadj* value of the calling process. The **exit** subroutine adds the *Semadj* value to the semval of the semaphore when the process terminates.

SEM_ORDER — Specifies whether to perform the operations specified by the *SemaphoreOperations* array of the **semop** subroutine atomically or individually. (This value is valid only when included in the *SemaphoreOperations*[0].sem_flg parameter, the first operation in the *SemaphoreOperations* array.)

If SEM_ORDER is not set (the default), the specified operations are performed atomically. That is, none of the semval values in the array are modified until all of the semaphore operations are completed. If the calling process must wait until some semval requirement is met, the **semop** subroutine does so before performing any of the operations. If any semaphore operation would cause an error to occur, none of the operations are performed.

If SEM_ORDER is set, the operations are performed individually in the order that they appear in the array, regardless of whether any of the operations require the process to wait. If an operation encounters an error condition, then the **semop** subroutine sets the SEM_ERR value in the sem_flg field of the failing operation; neither the failing operation nor the following operations in the array are performed.

IPC_NOWAIT — Specifies whether to wait or to return immediately when the semval of a semaphore is not a certain value.

The data type of the sem_flg field is short.

The **semid_ds** structure stores semaphore status information used by the **semctl** subroutine and pointed to by the *Buffer* parameter. This structure contains the following fields:

sem_perm — Specifies the operation permission structure of a semaphore. The data type of this field is struct ipc_perm.

sem_nsems — Specifies the number of semaphores in the set. The data type of this field is unsigned short.

sem_otime Specifies the time at which a **semop** subroutine was last performed. The data type of this field is time_t.

sem_ctime Specifies the time at which this structure was last changed with a **semctl** subroutine. The data type of this field is time_t.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/sys/sem.h The path to the **sem.h** header file.

Related Information

The **exec** subroutine, **exit** subroutine **fork** subroutine, **semctl** subroutine, **semget** subroutine, **semop** subroutine, **sigaction** subroutine.

The **exit** subroutine, **atexit** subroutine.

sgtty.h File

Purpose

Provides the terminal interface for the Berkeley line discipline.

Description

The **sgtty.h** header file defines the structures that are used by **ioctl** subroutines that apply to terminal files. The structures, definitions, and values in this file are provided for compatibility with the Berkeley user interface for asynchronous communication. Window and terminal size operations use the **winsize** structure, which is defined in the **ioctl.h** header file. The **winsize** structure and the **ioctl** functions that use it are described in the discussion of TTY common code in the TTY Subsystem Overview.

Note: Some documentation refers to the terminal drivers that are supported in the Berkeley interface as line disciplines. However, the term *line disciplines* in this documentation is used in a broader sense to refer to the entire asynchronous communications user interface (the Berkeley and POSIX line disciplines).

Basic sgtty.h Modes

The basic **ioctl** functions use the **sgttyb** structure defined in the **sgtty.h** header file. This structure contains the following fields:

sg_ispeed — Specifies the input speed of the device. For any particular hardware, impossible speed changes are ignored. Symbolic values in the table are as defined in the **sgtty.h** file.

B0	Hangs up. The zero baud rate is used to hang up the connection. If B0 is specified, the 'data terminal ready' signal is dropped. Normally, this disconnects the line.
B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
EXTA	External A
EXTB	External B

sg_ospeed — Specifies the output speed of the device. Refer to the description of the sg_ispeed field. ospeed has the same values as ispeed.

sg_erase — Specifies the erase character. (The default is Backspace.)

sg_kill — Specifies the kill character. (The default is Ctrl–U.)

sg_flags — Specifies how the system treats output. The initial output control value is all bits clear. The possible output modes are described in the following list.

ALLDELAY — Delay algorithm selection.

BSDELAY — Select backspace delays. Backspace delays are currently ignored. Possible values are BS0 or BS1.

VTDELAY — Select form-feed and vertical-tab delays:

- FF0 — Specifies no delay.
- FF1 — Specifies one delay of approximately 2 seconds.

CRDELAY — Select carriage-return delays:

- CR0 — Specifies no delay.
- CR1 — Specifies one delay. The delay lasts approximately .08 seconds.
- CR2 — Specifies one delay. The delay lasts approximately .16 seconds.
- CR3 — Specifies one delay. The delay pads lines to be at least 9 characters at 9600 baud.

TBDELAY — Select tab delays:

- TAB0 — Specifies no delay.
- TAB1 — Specifies one delay. The delay is dependent on the amount of movement.
- TAB2 — Specifies one delay. The delay lasts about .10 seconds.
- XTABS — Specifies that tabs are to be replaced by the appropriate number of spaces on output.

NLDELAY — Selects the new-line character delays. This is a mask to use before comparing to NL0 and NL1.

- NL0 — Specifies no delay.
- NL1 — Specifies one delay. The delay is dependent on the current column.
- NL2 — Specifies one delay. The delay lasts about .10 seconds.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. The actual delays depend on line speed and system load.

EVENP — Allows even parity on input.

The flags for even and odd parity control parity checking on input and generation on output in cooked and CBREAK mode (unless LPASS8 is enabled). Even parity is generated on output unless ODDP is set and EVENP is clear, in which case odd parity is generated. Input characters with the wrong parity, as determined by the EVENP and ODDP flags, are ignored in cooked and CBREAK mode.

ODDP — Allows odd parity on input. Refer to the description of the EVENP flag.

RAW — Indicates the RAW mode which features a wake up on all characters and an 8–bit interface.

The RAW mode disables all processing except output flushing with LFLUSHO. The full 8 bits of input are given as soon as they are available; all 8 bits are passed on output. A break condition in the input is reported as a null character. If the input queue overflows in RAW mode, all data in the input and output queues are discarded; this applies to both the new and old drivers.

CRMOD	Maps a carriage return into a new-line on input and outputs a new-line as a carriage return and a new-line.
ECHO	Echo (full duplex).
LCASE	Map uppercase to lowercase on input and lowercase to uppercase on output on uppercase terminals.
CBREAK	Enables a half-cooked mode. Programs can read each character as it is typed instead of waiting for a full line. All processing is done except the input editing. Character and word erase, line kill, input reprint, and special treatment of the backslash character and EOT are disabled.
TANDEM	Enables automatic flow control (the TANDEM mode), which causes the system to produce a stop character (Ctrl–S) whenever the input queue is in danger of overflowing, and a start character (Ctrl–Q) when the input queue has drained sufficiently. This mode is useful for flow control when the terminal is actually another computer that understands the conventions.

Note: The same stop and start characters are used for both directions of flow control; the t_stopc character is accepted on input as the character that stops output and is produced on output as the character to stop input, and the t_startc character is accepted on input as the character that restarts output and is produced on output as the character to restart input.

Basic ioctls

A large number of **ioctl** commands apply to terminals. Some have the general form:

```
#include <sgtty.h>
ioctl(FileDescriptor, Code, Value)
struct sgttyb *Value;
```

The applicable values for the *Code* parameter are:

TIOCGETP	Fetch the basic parameters associated with the terminal, and store in the pointed-to **sgttyb** structure.
TIOCSETP	Set the parameters according to the pointed-to **sgttyb** structure. The interface delays until output stops, then throws away any unread characters before changing the modes.
TIOCSETN	Set the parameters like TIOCSETP but do not delay or flush input. Input is not preserved, however, when changing to or from the RAW mode.

With the following codes, the *Value* parameter is ignored.

TIOCEXCL Set exclusive-use mode; no further opens are permitted until the file is closed.

TIOCNXCL Turn off exclusive-use mode.

TIOCHPCL When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with a modem used to place outgoing calls.

With the following codes, the *Value* parameter is a pointer to an integer.

TIOCGETD The *Value* parameter is a pointer to an integer into which is placed the current terminal driver number. The integer is OTTYDISC for the old Berkeley terminal driver and NTTYDISC for the standard (new) driver.

TIOCSETD The *Value* parameter is a pointer to an integer whose value becomes the current terminal driver number.

TIOCFLUSH If the integer pointed to by the *Value* parameter has a zero value, all characters waiting in input or output queues are flushed. Otherwise, the value of the integer is for the FREAD and FWRITE bits defined in the **file.h** header file; if the FREAD bit is set, all characters waiting in input queues are flushed, and if the FWRITE bit is set, all characters waiting in output queues are flushed.

In the following codes, the argument is 0 unless specified otherwise.

TIOCSTI The *Value* parameter points to a character that the system pretends had been typed on the terminal.

TIOCSBRK The break bit is set in the terminal.

TIOCCBRK The break bit is cleared.

TIOCSDTR Data terminal ready is set.

TIOCCDTR Data terminal ready is cleared.

TIOCSTOP Output is stopped as if the stop character had been typed.

TIOCSTART Output is restarted as if the start character had been typed.

TIOCGPGRP The *Value* parameter is a pointer to an integer into which is placed the process group ID of the process group for which this terminal is the control terminal.

TIOCSPGRP The *Value* parameter is a pointer to an integer which is the value to which the process group ID for this terminal will be set.

TIOCOUTQ Returns in the integer pointed to by the *Value* parameter the number of characters queued for output to the terminal.

FIONREAD Returns in the integer pointed to by the *Value* parameter the number of characters immediately readable from the argument descriptor. This works for files, pipes, and terminals.

Uppercase Terminals

If the LCASE output mode is set, then all uppercase letters are mapped into the corresponding lowercase letter. The uppercase letter can be generated by preceding it with a backslash (\). Uppercase letters are preceded by a backslash when they are output. In addition, the following escape sequences can be generated on output and accepted on input:

For:	Use:
` (grave)	\'
|	\!
~	\^
{	\(
}	\)

To deal with terminals that do not understand that the tilde (~) has been made into an ASCII character, the LTILDE bit can be set in the local mode word. When the LTILDE bit is set, the tilde character will be replaced with the grave character (`) on output.

Special Characters

A **tchars** structure associated with each terminal specifies special characters for both the old and new terminal interfaces. This structure is defined in the **ioctl.h** header file, for which the **sgtty.h** header file contains an **#include** statement. The **tchars** structure contains the following fields:

t_intrc — The interrupt character (Ctrl–C by default) generates a **SIGINT** signal. This is the normal way to stop a process which is no longer needed or to regain control in an interactive program.

t_quitc — The quit character (Ctrl–\ by default) generates a **SIGQUIT** signal. This is used to end a program and produce a core image, if possible, in a **core** file in the current directory.

t_startc — The start output character (Ctrl–Q by default).

t_stopc — The stop output character (Ctrl–S by default).

t_eofc — The end-of-file character (Ctrl–D by default).

t_brkc — The input delimiter (–1 by default). This character acts like a new-line in that it ends a line, is echoed, and is passed to the program.

The stop and start characters can be the same to produce a toggle effect. The applicable **ioctl** functions are:

TIOCGETC — Get the special characters and put them in the specified structure.

TIOCSETC — Set the special characters to those given in the structure.

Local Mode

Associated with each terminal is a local mode word. The bits of the local mode word are:

LCRTBS	Backspace on erase rather than echoing erase.
LPRTERA	Printing terminal erase mode.
LCRTERA	Erase character echoes as backspace–space–backspace.
LTILDE	Convert ~ to ' on output (for terminals that do not recognize the tilde as an ASCII character).
LMDMBUF	Stop and start output when carrier drops.
LLITOUT	Suppress output translations.
LTOSTOP	Send a **SIGTTOU** signal for background output.
LFLUSHO	Output is being flushed.
LNOHANG	Do not send hang up when carrier drops.
LCRTKIL	Backspace–space–Backspace to erase the entire line on line kill.
LPASS8	Pass all 8 bits through on input, in any mode.
LCTLECH	Echo input control characters as ^X, delete as ^?.
LPENDIN	Retype pending input at next read or input character.
LDECCTQ	Only Ctrl–Q restarts output after a Ctrl–S.
LNOFLSH	Inhibit flushing of pending I/O when an interrupt character is typed.

The following **ioctl** functions operate on the local mode word structure:

TIOCLBIS	The *Value* parameter is a pointer to an integer whose value is a mask containing the bits to be set in the local mode word.
TIOCLBIC	The *Value* parameter is a pointer to an integer whose value is a mask containing the bits to be cleared in the local mode word.
TIOCLSET	The *Value* parameter is a pointer to an integer whose value is stored in the local mode word.
TIOCLGET	The *Value* parameter is a pointer to an integer into which the current local mode word is placed.

Local Special Characters

The **ltchars** structure associated with each terminal defines control characters for the new terminal driver. This structure contains the following fields:

t_suspc	The suspend process character (Ctrl–Z by default). This sends a **SIGTSTP** signal to suspend the current process group. This character is recognized during input.
t_dsuspc	The delayed suspend process character (Ctrl–Y by default). This sends a **SIGTSTP** signal to suspend the current process group. This character is

recognized when the process attempts to read the control character rather than when it is typed.

t_rprntc — The reprint line control character (Ctrl–R by default). This reprints all characters that are preceded by a new-line character and have not been read.

t_flushc — The flush output character (Ctrl–O by default). This flushes data that is written but not transmitted.

t_werasc — The word erase character (Ctrl–W by default). This erases the preceding word. This does not erase beyond the beginning of the line.

t_lnextc — The literal next character (Ctrl–V by default). This causes the special meaning of the next character to be ignored so that characters can be input without being interpreted by the system.

The following **ioctl** functions, which use the **ltchars** structure, are supported by the terminal interface for the definition of local special characters for a terminal:

TIOCSLTC — Set local characters. The argument to this function is a pointer to an **ltchars** structure, which defines the new local special characters.

TIOCGLTC — Set local characters. The argument to this function is a pointer to an **ltchars** structure into which is placed the current set of local special characters.

The **winsize** structure and the **ioctl** functions that use it are described in the discussion of the TTY common code in the TTY Subsystem Overview.

For general information about the TTY subsystem, refer to the TTY Subsystem Overview in *General Programming Concepts*. For general information about the Berkeley line discipline, refer to Understanding the Berkeley Line Discipline in *General Programming Concepts*.

Implementation Specifics

This file is for Berkeley compatibility.

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/include/sys/file.h — The path to the **file.h** header file, which defines the FREAD and FWRITE bits used by the TIOCFLUSH **ioctl** command. The **sgtty.h** header file contains an **#include** statement for the **file.h** header file.

/usr/include/sys/ioctl.h — The path to the **ioctl.h** header file, which defines several of the ioctls used by the **sgtty.h** file. The **sgtty.h** header file contains an **#include** statement for the **ioctl.h** header file.

/dev/tty — The **tty** special file, which is a synonym for the controlling terminal.

/usr/include/sgtty.h — The path to the **sgtty.h** header file.

Related Information

The **csh** command, **getty** command, **stty** command, **tset** command.

The **ioctl** subroutine, **sigvec** subroutine.

srcobj.h File

Purpose

Defines structures used by the System Resource Controller (SRC) subsystem.

Description

The **srcobj.h** header file contains the **SRCsubsys** structure, which contains the following fields:

subsysname	The string that contains the subsystem name. The string can contain 14 characters.
synonym	The string that contains the subsystem synonym.The string can contain 14 characters.
cmdargs	The string that contains the subsystem command arguments.The string can contain 99 characters.
path	The string that contains the path to the executable files.The string can contain 99 characters.
uid	The user ID for the subsystem.
auditid	The audit ID for the subsystem. This value is supplied by the system and cannot be changed by an SRC subroutine.
standin	The string that contains the path for standard input.The string can contain 99 characters.
standout	The string that contains the path for standard output.The string can contain 99 characters.
standerr	The string that contains the path for standard error.The string can contain 99 characters.
action	Respawn action. The value of this field can be either `ONCE` or `RESPAWN`.
multi	Multiple instance support. The value of this field can be either `SRCYES` or `SRCNO`.
contact	Contact type. The value of this field indicates either a signal (`SRCSIGNAL`), a message queue (`SRCIPC`), or a socket (`SRCSOCKET`).
srvkey	The IPC message queue key.
svrmtype	The IPC mtype for the subsystem.
priority	Nice value, a number from 1 to 40.
signorm	Stop normal signal.
sigforce	Stop force signal.
display	Display inactive subsystem on all or group status. The value of this field can be either `SRCYES` or `SRCNO`.

waittime — Stop cancel time to wait before sending a **SIGKILL** signal to the subsystem restart time period. (A subsystem can be restarted only twice in this time period if it does not terminate normally.

grpname — The string that contains the group name of the subsystem.The string can contain 14 characters.

The **srcobj.h** header file also contains the **SRCsubsvr** structure, which contains the following fields:

sub_type — The string that contains the type of the subsystem. The string can contain 14 characters.

subsysname — The string that contains the subsystem name. The string can contain 14 characters.

sub_code — The string that contains the subsystem code. The string can contain 99 characters.

The size specified for the strings in the **SRCsubsys** structure does not include the null character that ends each of the strings. The possible values indicated for the strings are predefined.

File

/usr/include/sys/srcobj.h — The path to the **srcobj.h** header file.

Related Information

The **getssys** subroutine.

stat.h File

Purpose

Defines the data structure returned by the status subroutines.

Description

The **stat** data structure in the **stat.h** header file is used to return information for the **stat, fstat, lstat, statx**, and **fstatx** subroutines.

The **stat** data structure contains the following fields:

st_dev — The device that contains a directory entry for this file.

st_ino — The index of this file on its device. A file is uniquely identified by specifying the device on which it resides and its index on that device.

st_mode — The file mode. The possible file mode values are given in the description of the **mode.h** header file.

st_nlink — The number of hard links (alternate directory entries) to the file created using the **link** subroutine.

st_access — The access that the calling process would be granted to the file. This is some combination of the IROTH (other read permission), IWOTH (other write permission), and IXOTH (other execute or search permission) bits.

st_size — The number of bytes in a file (including any holes). This field also defines the position of the end-of-file mark for the file. The end-of-file mark is updated only by subroutines, for example the **write** subroutine. If the file is mapped by the **shmat** subroutine and a value is stored into a page past the end-of-file mark, the end-of-file mark will be updated to include this page when the file is closed or forced to permanent storage.

st_rdev — The ID of the device. This field is defined only for block or character special files.

st_atime — The time when file data was last accessed.

st_mtime — The time when file data was last modified.

st_ctime — The time when the file status was last changed.

st_blksize — The size, in bytes of each block of the file.

st_blocks — The number of blocks actually used by the file (measured in the units specified by the DEV_BSIZE constant).

st_gen — The generation number of this i-node.

st_type — The type of the v-node for the object. This is one of the following values, which are defined in the **vnode.h** header file:

VNON	An unallocated object; this should not occur.
VBAD	An unknown type of object.
VREG	A regular file.

VDIR	A directory file.
VBLK	A block device.
VCHR	A character device.
VLNK	A symbolic link.
VSOCK	A socket.
VFIFO	FIFO.
VMPC	A multiplexed character device.

st_vfs — Virtual file system (VFS) ID, which identifies the VFS that contains the file. By comparing this value with the VFS numbers returned by the **mntctl** subroutine, the name of the host where the file resides can be identified.

st_vfstype — File system type, as defined in the **vmount.h** header file.

st_flag — A flag indicating whether the file or the directory is a virtual mount point. This flag can have the following values:

FS_VMP	Indicates that the file is a virtual mount point.
FS_MOUNT	Indicates that the file is a virtual mount point.
FS_REMOTE	Indicates that the file resides on another machine.

st_uid — The file owner ID.

st_gid — The file group ID.

For remote files, the st_atime, st_mtime, and st_ctime fields contain the time at the server.

The value of the st_atime field can be changed by the following subroutines:

read, readx, readv, readvx
readlink
shmdt
utime, utimes

The values of the st_ctime field and st_mtime field can be set by the following subroutines:

write, writex, writev, writevx
open, openx, creat
link
symlink
unlink
mknod
mkdir
rmdir
rename
truncate, ftruncate
utime, utimes

In addition, the **shmdt** subroutine can change the st_mtime field and the **chmod, fchmod, chown, chownx, fchown,** and **fchownx** subroutine can change the st_ctime field.

Because they can create a new object, the **open, openx, creat, symlink, mknod, mkdir,** and **pipe** subroutines can set the st_atime, st_ctime, and st_mtime fields.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of the AIX Base Operating System (BOS) Runtime.

Files

/usr/include/sys/stat.h	The path to the **stat.h** header file.
/usr/include/sys/mode.h	The path to the **mode.h** header file, which contains definitions of the file mode values.
/usr/include/sys/vmount.h	The path to the **vmount.h** header file, which describes the structure of a virtual file system.
/usr/include/sys/vnode.h	The path to the **vnode.h** header file, which contains definitions of the v-node types.

Related Information

The **chmod** subroutine, **chownx** subroutine, **fstatx** subroutine, **link** subroutine, **mknod** subroutine, **openx** subroutine, **pipe** subroutine, **read** subroutine, **shmat**, **statx** subroutine, **unlink**, **utime** subroutine, **write** subroutine.

The **mode.h** file, **types.h** file, **vmount.h** file.

statfs.h File

Purpose

Describes the structure of the statistics returned by the **statfs, fstatfs, ustat** subroutines.

Description

The **statfs** and **fstatfs** subroutines return information on a mounted (virtual) file system in the form of a **statfs** structure. The **statfs.h** header file describes the **statfs** structure, which contains the following fields:

f_version	The version number of the **statfs** structure. This value is currently 0.
f_length	The length of the buffer that contains the returned information. This value is currently 0.
f_type	The type of information returned. This value is currently 0.
f_bsize	The size of the fundamental file system.
f_blocks	The total number of blocks in the system.
f_bfree	The number of free blocks in the file system.
f_bavail	The number of free blocks that are available to a non-root user.
f_files	The total number of file nodes in the file system.
f_ffree	The number of free file nodes in the file system.
f_fsid	The file system ID.
f_vfstype	The type of this virtual file system.
f_fname[32]	The file system name (mount point).
f_fpack[32]	The file system pack name.
f_name_max	The maximum component name for this file system

Fields that are not defined for a particular file system are set to a value of –1.

The **ustat** system returns information on a mounted file system in the form of a **ustat** structure. The **ustat** structure, which is defined in the **ustat.h** header file, contains the following fields:

f_tfree	The total number of free blocks in the file system.
f_inode	The number of free i-nodes in the file system.
f_fname[6]	The file system name.
f_fpack[6]	The file system pack name.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

Files

/usr/include/sys/statfs.h	The path to the **statfs.h** header file.
/usr/include/ustat.h	The path to the **ustat.h** header file.

Related Information

The **statfs, fstatfs,** or **ustat** subroutine.

termio.h File

Purpose

Defines the structure of the **termio** file, which provides the terminal interface for AIX Version 2 compatibility.

Description

The **termio.h** header file contains the **termio** structure, which defines special characters as well as the basic input, output, control, and line discipline modes. The **termio.h** header file is provided for compatibility with AIX Version 2 applications.

AIX Version 2 applications that include the **termio.h** header file can use the AIX Version 2 terminal interface provided by the POSIX line discipline. The following AIX Version 2 terminal interface operations are not supported by the POSIX line discipline:

- Terminal Paging (TCGLEN **ioctl** and TCSLEN **ioctl**)
- Terminal Logging (TCLOG **ioctl**)
- Enhanced Edit Line Discipline (LDSETDT **ioctl** and LDGETDT **ioctl**)

The **termio** structure in the **termio.h** header file contains the following fields:

c_iflag — Describes the basic terminal input control. The initial input control value is all bits clear. The possible input modes are:

IGNBRK — Ignores the break condition. In the context of asynchronous serial data transmission, a *break condition* is defined as a sequence of zero-valued bits that continues for more than the time required to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for an amount of time equivalent to more than one byte. If IGNBRK is set, a break condition detected on input is ignored, which means that it is not put on the input queue and therefore not read by any process.

BRKINT — Signal interrupt on the break condition. If IGNBRK is not set and BRKINT is set, the break condition flushes the input and output queues. If the terminal is the controlling terminal of a foreground process group, the break condition generates a single **SIGINT** signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single \0; if PARMRK is set, a break condition is read as \377, \0, \0.

IGNPAR — Ignores characters with parity errors. If set, a byte with a framing or parity error (other than break) is ignored.

PARMRK — Marks parity errors. If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence \377, \0, *x*, where \377, \0 is a two-character flag preceding each sequence and *x* is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of \377 is given to the application as \377, \377. If neither IGNPAR nor PARMRK is set, a framing or

parity error (other than break) is given to the application as a single character \0.

INPCK — Enables input parity checking. If set, input parity checking is enabled. If not set, input parity checking is disabled. This allows for output parity generation without input parity errors.

ISTRIP — Strips characters. If set, valid input characters are first stripped to 7 bits; otherwise all 8 bits are processed.

INLCR — Maps new-line character (NL) to carriage return (CR) on input. If set, a received NL character is translated into a CR character.

IGNCR — Ignores CR character. If set, a received CR character is ignored (not read).

ICRNL — Maps CR character to NL character on input. If ICRNL is set and IGNCR is not set, a received CR character is translated into a NL character.

IUCLC — Maps uppercase to lowercase on input. If set, a received uppercase, alphabetic character is translated into the corresponding lowercase character.

IXON — Enables start and stop output control. If set, a received STOP character suspends output and a received START character restarts output. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters are read.

IXANY — Enables any character to restart output. If set, any input character restarts output that was suspended.

IXOFF — Enables start and stop input control. If set, the system transmits a STOP character when the input queue is nearly full and a START character when enough input has been read that the queue is nearly empty again.

c_oflag — Specifies how the system treats output. The initial output control value is all bits clear. The possible output modes are:

OPOST — Post-processes output. If set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

OLCUC — Maps lowercase to uppercase on output. If set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with the IUCLC input mode.

ONLCR — Maps NL to CR-NL on output. If set, the NL character is transmitted as the CR-NL character pair.

OCRNL — Maps CR to NL on output. If set, the CR character is transmitted as the NL character.

ONOCR — Indicates no CR output at column 0. If set, no CR character is transmitted when at column 0 (first position).

ONLRET — NL performs CR function. If set, the NL 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. If neither ONLCR, OCRNL, ONOCR, nor ONLRET is set, the NL character is assumed to do the line feed function only; the column pointer remains unchanged. The column pointer is also set to a value of 0 if the CR character is actually transmitted.

The delay bits specify how long a transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. The actual delays depend on line speed and system load.

OFILL — Uses fill characters for delay. If set, fill characters are transmitted for a delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay.

OFDEL — Sets fill characters to the DEL value. If set, the fill character is DEL. If this flag is not set, the fill character is NULL.

NLDLY — Selects the new-line character delays. This is a mask to use before comparing to NL0 and NL1.

- NL0 — Specifies no delay.
- NL1 — Specifies one delay of approximately 0.10 seconds. If ONLRET is set, the carriage return delays are used instead of the new-line delays. If OFILL is set, two fill characters are transmitted.

CRDLY — Selects the carriage return delays. This is a mask to use before comparing to CR0, CR1, CR2 and CR3.

- CR0 — Specifies no delay.
- CR1 — Specifies that the delay is dependent on the current column position. If OFILL is set, this delay transmits two fill characters.
- CR2 — Specifies one delay of approximately 0.10 seconds. If OFILL is set, this delay transmits four fill characters.
- CR3 — Specifies one delay of approximately 0.15 seconds.

TABDLY — Selects the horizontal-tab delays. This is a mask to use before comparing to TAB0, TAB1, TAB2, and TAB3. If OFILL is set, any of these delays (except TAB3) transmit two fill characters.

- TAB0 — Specifies no delay.
- TAB1 — Specifies that the delay is dependent on the current column position. If OFILL is set, two fill characters are transmitted.
- TAB2 — Specifies one delay of approximately 0.10 seconds.
- TAB3 — Specifies that tabs are to be expanded into spaces.

BSDLY — Selects the backspace delays. This is a mask to use before comparing to BS0, BS1.

BS0	Specifies no delay.
BS1	Specifies one delay of approximately 0.05 seconds. If OFILL is set, this delay transmits one fill character.

VTDLY — Selects the vertical-tab delays. This is a mask to use before comparing to VT0 and VT1.

VT0	Specifies no delay.
VT1	Specifies one delay of approximately 2 seconds.

FFDLY — Selects the form-feed delays. This is a mask to use before comparing to FF0 and FF1.

FF0	Specifies no delay.
FF1	Specifies one delay of approximately 2 seconds.

c_cflag — Describes the hardware control of the terminal. In addition to the basic control modes, this field uses the following control characters:

CBAUD — Specifies baud rate. These bits specify the baud rate for a connection. For any particular hardware, impossible speed changes are ignored.

B0	Hangs up. The zero baud rate is used to hang up the connection. If B0 is specified, the 'data terminal ready' signal is not asserted. Normally, this disconnects the line.
B50	50 baud.
B75	75 baud.
B110	110 baud.
B134	134.5 baud.
B150	150 baud.
B200	200 baud.
B300	300 baud.
B600	600 baud.
B600	600 baud.
B1200	1200 baud.
B1800	1800 baud.
B2400	2400 baud.
B4800	4800 baud.
B9600	9600 baud.
B19200	19200 baud.
B38400	38400 baud.
EXTA	External A.
EXTB	External B.

CSIZE — Specifies the character size. These bits specify the character size in bits for both transmit and receive operations. This size does not include the parity bit, if any.

CS5	5 bits.
CS6	6 bits.
CS7	7 bits.
CS8	8 bits.

CSTOPB — Specifies number of stop bits. If set, 2 stop bits are sent; otherwise, only 1 stop bit is sent.

CREAD — Enables receiver. If set, the receiver is enabled. Otherwise, characters are not received.

PARENB — Enables parity. If set, parity generation and detection is enabled and a parity bit is added to each character.

PARODD — Specifies odd parity. If parity is enabled, PARODD specifies odd parity if it is set. If parity is enabled and PARODD is not set, even parity is used.

HUPCL — Hangs up on last close. If set, the line is disconnected when the last process closes the line or when the process terminates (when the 'data terminal ready' signal drops).

CLOCAL — Specifies a local line. If set, the line is assumed to have a local, direct connection with no modem control. If not set, modem control (dial-up) is assumed.

c_lflag — Controls various terminal functions. The initial value after an open is all bits clear. This field uses the following mask name symbols:

ISIG — Enables signals. If set, each input character is checked against the INTR and QUIT special control characters. If an input character matches one of these control characters, the function associated with that character is performed. If the ISIG function is not set, checking is not done.

ICANON — Enables canonical input. If set, turns on canonical processing, which enables the erase and kill edit functions as well as the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. In this case, a read request is not satisfied until one of the following conditions is met: a) the minimum number of characters specified by MIN are received; or b) the time-out value specified by 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 — Enables canonical uppercase and lowercase presentation. If set along with ICANON, an uppercase letter (or the uppercase letter translated to lowercase by the IUCLC input mode) is accepted on input by preceding it with a \ (backslash) character. The output is then preceded by a backslash character. In this mode, the output generates and the input accepts the following escape sequences:

For:	Use:
` (grave)	\ `
|	\ !
~	\ ^
{	\ (
}	\ )
\	\ \

For example, A is input as \a, \n as \\n, and \N as \\\n.

NOFLSH — Disables queue flushing. If set, the normal flushing of the input and output queues associated with the INTR and QUIT characters is not done.

ECHO — Enables echo. If set, characters are echoed as they are received.

When ICANON is set, the following echo functions are possible:

ECHOE — Echoes the erase character as Backspace–space–Backspace. If ECHO and ECHOE are both set, the ERASE character is echoed as one or more ASCII Backspace–space–Backspace sequences, which clears the last character(s) from the screen.

ECHOK — Echoes NL after kill. If ECHOK is set, the NL character is echoed after the kill character is received. This emphasizes that the line is deleted.

ECHONL — Echoes NL. If ECHONL is set, the NL character is echoed even if ECHO is not set. This is useful for terminals that are set to local echo (also referred to as half-duplex).

c_cc — Specifies an array that defines the special control characters. The relative positions and initial values for each function are:

VINTR — Indexes the INTR special character (Ctrl–C), which is recognized on input if ISIG is set. The INTR character generates a SIGINT signal, which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character is discarded when processed.

VQUIT — Indexes the QUIT special character (Ctrl–\), which is recognized on input if ISIG is set. The QUIT character generates a SIGQUIT signal, which is sent to all processes in the foreground process group for which the terminal is the controlling terminal, and writes a **core** image file into the current working directory. If ISIG is set, the QUIT character is discarded when processed.

VERASE — Indexes the ERASE special character (Backspace), which is recognized on input if ICANON is set. The ERASE character does not erase beyond the beginning of the line as delimited by a NL, EOL, EOF, or EOL2 character. If ICANON is set, the ERASE character is discarded when processed.

VKILL — Indexes the KILL special character (Ctrl–U), which is recognized on input if ICANON is set. The KILL character deletes the entire line, as delimited by a NL, EOL, EOF, or EOL2 character. If ICANON is set, the KILL character is discarded when processed.

VEOF — Indexes the EOF special character (Ctrl–D), which is recognized on input if ICANON is set. When EOF is received, all the characters waiting to be read are immediately passed to the process, without waiting for a new-line, and the EOF is discarded. If the EOF is received at the beginning of a line (no characters are waiting), a character count of zero is returned from the read, indicating an end-of-file. If ICANON is set, the EOF character is discarded when processed.

VEOL — Indexes the EOL special character (Ctrl–@ or ASCII NULL), which is recognized on input if ICANON is set. EOL is an additional line delimiter, like NL. It is not normally used.

VEOL2 — Indexes the EOL2 special character (Ctrl–@ or ASCII NULL), which is recognized on input if ICANON is set. EOL2 is another additional line delimiter, like NL. It is not normally used.

VMIN — Indexes the MIN value, which is not a special character. The use of the MIN value is described in the discussion of non-canonical mode input processing in "Understanding the POSIX Line Discipline."

VTIME — Indexes the TIME value, which is not a special character. The use of the TIME value is described in the discussion of non-canonical mode input processing in "Understanding the POSIX Line Discipline."

The character values for the following control characters can be changed:

INTR	ERASE	EOF	EOL2
QUIT	KILL	EOL	

The ERASE, KILL, and EOF characters can also be escaped (preceded with a backslash) so that no special processing is done.

The primary **ioctl** subroutines have the form:

```
ioctl (FileDescriptor, Command, Structure) struct termio *Structure;
```

The commands using this form are:

TCGETA — Get the parameters associated with the terminal and store them in the **termio** structure referenced by the *Structure* parameter.

TCSETA — Set the parameters associated with the terminal from the structure referenced by the *Structure* parameter. The change is immediate.

TCSETAF — Wait for the output to drain, then flush the input queue and set the new parameters.

TCSETAW Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

Additional **ioctl** subroutines have the form:

```
ioctl (FileDescriptor, Command, Value) int Value;
```

The commands using this form are:

TCSBRK Wait for the output to drain. If *Value* is 0, then send a break of 0.25 seconds in duration. If *Value* is non-zero, cause a break condition of *Value* milliseconds in duration.

TCSBREAK Wait for the output to drain. If *Value* is 0, then send a break of .25 seconds in duration. If *Value* is non-zero, cause a break condition of *Value* milliseconds in duration.

TCXONC Start and stop control. If *Value* is 0, suspend output; if 1, restart suspended output; if 2, block intput; if 3, unblock input.

TCFLSH If *Value* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

The following **ioctl** operations are used for trusted communications path operations:

TCSAK Points to an integer that enables the Secure Attention Key (SAK) sequence (Ctrl–X Ctrl–R) in order to provide a clean terminal to which only trusted processes can read or write. When SAK is enabled and the user types this sequence, all processes that are currently running are ended. The TCSAKON operation turns on the SAK sequence; the TCSAKOFF operation turns off the SAK sequence.

TCQSAK Queries the state (TCSAKON or TCSAKOFF) of the SAK sequence.

TCTRUST Sets a bit by which another process can query (with the TCQTRUST operation) the state of the terminal, TCTRUSTED or TCUNTRUSTED.

TCQTRUST Queries the state of the terminal (TCTRUSTED or TCUNTRUSTED).

Also refer to the description of the **termios.h** header file.

For general information about the TTY subsystem, refer to TTY Subsystem Overview in *General Programming Concepts*. For general information about the AIX Version 2 compatibility mode, refer to Understanding AIX Version 2 Compatibility Mode in *General Programming Concepts*. For general information about the POSIX line discipline, refer to Understanding the POSIX Line Discipline in *General Programming Concepts*.

Files

/usr/include/sys/ioctl.h	The path to the **ioctl.h** header file, which contains the **winsize** structure and many of the ioctls used by this line discipline. The **termio.h** file contains an **#include** statement for the **ioctl.h** file.
/usr/include/sys/termio.h	The path to the **termio.h** header file.
/usr/include/termios.h	The path to the **termios.h** header file, which contains the structures used by the POSIX line discipline.

Implementation Specifics

This file is for compatibility with AIX Version 2.

This file is part of AIX Base Operating System (BOS) Runtime.

Related Information

The **fork** subroutine, **ioctl** subroutine, **setpgrp** subroutine, **sigvec** subroutine.

The **csh** command, **getty** command, **stty** command, **tset** command.

termios.h File

Purpose

Defines the structure of the **termios** file, which provides the terminal interface for POSIX compatibility.

Description

The **termios.h** header file contains information used by subroutines that apply to terminal files. The definitions, values, and structures in this file are required for compatibility with the Institute of Electrical and Electronics Engineers (IEEE) P1003.1 Portable Operating System Interface for Computer Environments (POSIX) standard.

The general terminal interface information is contained in the **termio.h** header file. The **termio** structure in the **termio.h** header file defines the basic input, output, control, and line discipline modes. If a calling program is identified as requiring POSIX compatibility, the **termios** structure and additional, POSIX control packet information in the **termios.h** header file is implemented. Window and terminal size operations use the **winsize** structure, which is defined in the **ioctl.h** header file.The **termios** structure in the **termios.h** header file contains the following fields:

c_iflag — Describes the basic terminal input control. The initial input control value is all bits clear. The possible input modes are:

- IGNBRK — Ignores the break condition. In the context of asynchronous serial data transmission, a *break condition* is defined as a sequence of zero-valued bits that continues for more than the time required to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for an amount of time equivalent to more than one byte. If IGNBRK is set, a break condition detected on input is ignored, which means that it is not put on the input queue and therefore not read by any process.
- BRKINT — Signal interrupt on the break condition. If IGNBRK is not set and BRKINT is set, the break condition flushes the input and output queues. If the terminal is the controlling terminal of a foreground process group, the break condition generates a single **SIGINT** signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single \0, or if PARMRK is set, as \377, \0, \0..
- IGNPAR — Ignores characters with parity errors. If set, a byte with a framing or parity error (other than break) is ignored.
- PARMRK — Marks parity errors. If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence \377, \0, *x*, where \377, \0 is a two-character flag preceding each sequence and *x* is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of \377 is given to the application as \377, \377. If neither IGNPAR nor PARMRK is set, a framing or parity error (other than break) is given to the application as a single character \0.

INPCK	Enables input parity checking. If set, input parity checking is enabled. If not set, input parity checking is disabled. This allows for output parity generation without input parity errors.
ISTRIP	Strips characters. If set, valid input characters are first stripped to 7 bits; otherwise all 8 bits are processed.
INLCR	Maps new-line character (NL) to carriage return (CR) on input. If set, a received NL character is translated into a CR character.
IGNCR	Ignores CR character. If set, a received CR character is ignored (not read).
ICRNL	Maps CR character to NL character on input. If ICRNL is set and IGNCR is not set, a received CR character is translated into a NL character.
IUCLC	Maps uppercase to lowercase on input. If set, a received uppercase, alphabetic character is translated into the corresponding lowercase character.
IXON	Enables start and stop output control. If set, a received STOP character suspends output and a received START character restarts output. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters are read.
IXANY	Enables any character to restart output. If set, any input character restarts output that was suspended.
IXOFF	Enables start and stop input control. If set, the system transmits a STOP character when the input queue is nearly full and a START character when enough input has been read that the queue is nearly empty again.
IMAXBEL	Echoes the ASCII BEL character if the input stream overflows. Further input is not stored, but any input present in the input stream is not lost. If not set, no BEL character is echoed; the input in the input queue is discarded if the input stream overflows.

c_oflag — Specifies how the system treats output. The initial output control value is all bits clear. The possible output modes are:

OPOST	Post-processes output. If set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.
OLCUC	Maps lowercase to uppercase on output. If set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with the IUCLC input mode.
ONLCR	Maps NL to CR-NL on output. If set, the NL character is transmitted as the CR-NL character pair.
OCRNL	Maps CR to NL on output. If set, the CR character is transmitted as the NL character.
ONOCR	Indicates no CR output at column 0. If set, no CR character is transmitted when at column 0 (first position).

ONLRET NL performs CR function. If set, the NL 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. If neither ONLCR, OCRNL, ONOCR, nor ONLRET is set, the NL character is assumed to do the line feed function only; the column pointer remains unchanged. The column pointer is also set to a value of 0 if the CR character is actually transmitted.

The delay bits specify how long a transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. The actual delays depend on line speed and system load.

OFILL Uses fill characters for delay. If set, fill characters are transmitted for a delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay.

OFDEL Sets fill characters to the DEL value. If set, the fill character is DEL. If this flag is not set, the fill character is NULL.

NLDLY Selects the new-line character delays. This is a mask to use before comparing to NL0 and NL1.

- NL0 Specifies no delay.
- NL1 Specifies one delay of approximately 0.10 seconds. If ONLRET is set, the carriage return delays are used instead of the new-line delays. If OFILL is set, two fill characters are transmitted.

CRDLY Selects the carriage return delays. This is a mask to use before comparing to CR0, CR1, CR2 and CR3.

- CR0 Specifies no delay.
- CR1 Specifies that the delay is dependent on the current column position. If OFILL is set, this delay transmits two fill characters.
- CR2 Specifies one delay of approximately 0.10 seconds. If OFILL is set, this delay transmits four fill characters.
- CR3 Specifies one delay of approximately 0.15 seconds.

TABDLY Selects the horizontal-tab delays. This is a mask to use before comparing to TAB0, TAB1, TAB2, and TAB3. If OFILL is set, any of these delays (except TAB3) transmit two fill characters.

- TAB0 Specifies no delay.
- TAB1 Specifies that the delay is dependent on the current column position. If OFILL is set, two fill characters are transmitted.
- TAB2 Specifies one delay of approximately 0.10 seconds.
- TAB3 Specifies that tabs are to be expanded into spaces.

BSDLY — Selects the backspace delays. This is a mask to use before comparing to BS0, BS1.

- BS0 Specifies no delay.
- BS1 Specifies one delay of approximately 0.05 seconds. If OFILL is set, this delay transmits one fill character.

VTDLY — Selects the vertical-tab delays. This is a mask to use before comparing to VT0 and VT1.

- VT0 Specifies no delay.
- VT1 Specifies one delay of approximately 2 seconds.

FFDLY — Selects the form-feed delays. This is a mask to use before comparing to FF0 and FF1.

- FF0 Specifies no delay.
- FF1 Specifies one delay of approximately 2 seconds.

c_cflag — Describes the hardware control of the terminal. In addition to the basic control modes, this field uses the following control characters:

CBAUD — Specifies baud rate. These bits specify the baud rate for a connection. For any particular hardware, impossible speed changes are ignored.

- B0 Hangs up. The zero baud rate is used to hang up the connection. If B0 is specified, the 'data terminal ready' signal is not asserted. Normally, this disconnects the line.
- B50 50 baud.
- B75 75 baud.
- B110 110 baud.
- B134 134.5 baud.
- B150 150 baud.
- B200 200 baud.
- B300 300 baud.
- B600 600 baud.
- B600 600 baud.
- B1200 1200 baud.
- B1800 1800 baud.
- B2400 2400 baud.
- B4800 4800 baud.
- B9600 9600 baud.
- B19200 19200 baud.
- B38400 38400 baud.
- EXTA External A.
- EXTB External B.

CSIZE — Specifies the character size. These bits specify the character size in bits for both transmit and receive operations. This size does not include the parity bit, if any.

- CS5 5 bits.
- CS6 6 bits.
- CS7 7 bits.
- CS8 8 bits.

CSTOPB — Specifies number of stop bits. If set, 2 stop bits are sent; otherwise, only 1 stop bit is sent.

CREAD — Enables receiver. If set, the receiver is enabled. Otherwise, characters are not received.

PARENB — Enables parity. If set, parity generation and detection is enabled and a parity bit is added to each character.

PARODD — Specifies odd parity. If parity is enabled, PARODD specifies odd parity if it is set. If parity is enabled and PARODD is not set, even parity is used.

HUPCL — Hangs up on last close. If set, the line is disconnected when the last process closes the line or when the process terminates (when the 'data terminal ready' signal drops).

CLOCAL — Specifies a local line. If set, the line is assumed to have a local, direct connection with no modem control. If not set, modem control (dial-up) is assumed.

CIBAUD — Specifies the input baud rate if it is different than the output rate.

PAREXT — Specifies extended parity for mark and space parity.

c_lflag — Controls various terminal functions. The initial value after an open is all bits clear. In addition to the basic modes, this field uses the following mask name symbols:

ISIG — Enables signals. If set, each input character is checked against the INTR, QUIT, SUSP, and DSUSP special control characters. If an input character matches one of these control characters, the function associated with that character is performed. If the ISIG function is not set, checking is not done.

ICANON — Enables canonical input. If set, turns on canonical processing, which enables the erase and kill edit functions as well as the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. In this case, a read request is not satisfied until one of the following conditions is met: a) the minimum number of characters specified by MIN are received; or b) the time-out value specified by 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 — Enables canonical uppercase and lowercase presentation. If set along with ICANON, an uppercase letter (or the uppercase letter translated to lowercase by the IUCLC input mode) is accepted on input by preceding it with a \ (backslash) character. The output is then preceded by a backslash character. In this mode, the output generates and the input accepts the following escape sequences:

For:	Use:
` (grave)	\ `
|	\ !
~	\ ^
{	\ (
}	\ )
\	\ \

For example, A is input as \a, \n as \\n, and \N as \\\n.

NOFLSH — Disables queue flushing. If set, the normal flushing of the input and output queues associated with the INTR, QUIT, and SUSP characters is not done.

FLUSHO — Flushes the output. When this bit is set by typing the FLUSH character, data written to the terminal is discarded. A terminal can cancel the effect of typing the FLUSH character by clearing this bit.

PENDIN — Reprints any input that has not yet been read when the next character arrives as input.

IEXTEN — Enables extended (implementation-defined) functions to be recognized from the input data. If this bit is not set, implementation-defined functions are not recognized, and the corresponding input characters are processed as described for ICANON, ISIG, IXON, and IXOFF.

TOSTOP — Sends a **SIGTTOU** signal when a process in a background process group tries to write to its controlling terminal. The **SIGTTOU** signal stops the members of the process group.

ECHO — Enables echo. If set, characters are echoed as they are received.

When ICANON is set, the following echo functions are also possible:

ECHOE — Echoes the erase character as Backspace–space–Backspace. If ECHO and ECHOE are both set and ECHOPRT is not set, the ERASE and WERASE characters are echoed as one or more ASCII Backspace–space–Backspace sequences, which clears the last character(s) from the screen.

ECHOPRT — If ECHO and ECHOPRT are set, echoes the first ERASE and WERASE character in a sequence as a backslash (\), followed by the characters being erased. Subsequent ERASE and WERASE characters echo the characters being erased, in reverse order. The next non-erase character causes a slash (/) to be typed before the non-erase character is echoed.

ECHOKE — Backspace–space–Backspace entire line on line kill. If set, the kill character is echoed by erasing the entire line from the screen (using the mechanism selected by ECHOE and ECHOPRT).

ECHOK — Echoes NL after kill. If ECHOK is set and ECHOKE is not set, the NL character is echoed after the kill character is received. This emphasizes that the line is deleted.

ECHONL — Echoes NL. If ECHONL is set, the NL character is echoed even if ECHO is not set. This is useful for terminals that are set to local echo (also referred to as half-duplex).

ECHOCTL — Echoes control characters (with codes between 0 and 37 octal) as ^*X*, where *X* is the character given by adding 100 octal to the code of the control character. (For example, the character with octal code 1 is echoed as ^A). The ASCII DEL character (code 177 octal) is echoed as ^?. The ASCII TAB, NL, and START characters are not echoed. 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.

c_cc — Specifies an array that defines the special control characters. The relative positions and initial values for each function are:

VINTR — Indexes the INTR special character (Ctrl–c), which is recognized on input if ISIG is set. The INTR character generates a SIGINT signal, which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character is discarded when processed.

VQUIT — Indexes the QUIT special character (Ctrl–\), which is recognized on input if ISIG is set. The QUIT character generates a SIGQUIT signal, which is sent to all processes in the foreground process group for which the terminal is the controlling terminal, and writes a **core** image file into the current working directory. If ISIG is set, the QUIT character is discarded when processed.

VERASE — Indexes the ERASE special character (Backspace), which is recognized on input if ICANON is set. The ERASE character does not erase beyond the beginning of the line as delimited by a NL, EOL, EOF, or EOL2 character. If ICANON is set, the ERASE character is discarded when processed.

VKILL — Indexes the KILL special character (Ctrl–u), which is recognized on input if ICANON is set. The KILL character deletes the entire line, as delimited by a NL, EOL, EOF, or EOL2 character. If ICANON is set, the KILL character is discarded when processed.

VEOF — Indexes the EOF special character (Ctrl–d), which is recognized on input if ICANON is set. When EOF is received, all the characters waiting to be read are immediately passed to the process, without waiting for a

	new-line; the EOF is discarded. If the EOF is received at the beginning of a line (no characters are waiting), a character count of zero is returned from the read, indicating an end-of-file. If ICANON is set, the EOF character is discarded when processed.
VEOL	Indexes the EOL special character (Ctrl–@ or ASCII NULL), which is recognized on input if ICANON is set. EOL is an additional line delimiter, like NL. It is not normally used.
VEOL2	Indexes the EOL2 special character (Ctrl–@ or ASCII NULL), which is recognized on input if ICANON is set. EOL2 is another additional line delimiter, like NL. It is not normally used.
VSTART	Indexes the START special character (Ctrl–q), which is recognized on input if IXON is set and generated on output if IXOFF is set. START can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character is discarded when processed. While output is not suspended, START characters are ignored and not read. VSTRT is an alias for VSTART.
VSTOP	Indexes the STOP special character (Ctrl–s), which is recognized on input if IXON is set and generated on output if IXOFF is set. STOP is useful with terminals to prevent output from disappearing before it can be read. If IXON is set, the Stop character is discarded when processed. While output is suspended, STOP characters are ignored and not read..
VSUSP	Indexes the SUSP special character (Ctrl–z), which is recognized on input if ISIG is set. The SUSP character generates a SIGTSTP signal, which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the SUSP character is discarded when processed.
VDSUSP	Indexes the DSUSP special character (Ctrl–y), which is recognized on input if ISIG is set. DSUSP generates a SIGTSTP signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when DSUSP is typed. If ISIG is set, the DSUSP character is discarded when processed.
VREPRINT	Indexes the REPRINT special character (Ctrl–r), which is recognized on input if ICANON is set. The REPRINT character reprints all characters, preceded by a new-line, that have not been read. If ICANON is set, the REPRINT character is discarded when processed.
VDISCRD	Indexes the DISCARD special character (Ctrl–o), which is recognized on input if ICANON is set. The DISCARD character causes subsequent output to be discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program. If ICANON is set, the DISCARD character is discarded when processed.

VWERSE — Indexes the WERASE special character (Ctrl–w), which is recognized on input if ICANON is set. The WERASE character causes the preceding word to be erased. The WERASE character does not erase beyond the beginning of the line as delimited by a NL, EOL, EOF, or EOL2 character. If ICANON is set, the WERASE character is discarded when processed.

VLNEXT — Indexes the LNEXT (literal next) special character (Ctrl–v), which is recognized on input if ICANON is set. The LNEXT character causes the special meaning of the next character to be ignored so that characters can be input without being interpreted by the system. If ICANON and ECHO are set, the LNEXT character is replaced by a ^-Backspace sequence when processed.

VMIN — Indexes the MIN value, which is not a special character. The use of the MIN value is described in the discussion of non-canonical mode input processing in Understanding the POSIX Line Discipline.

VTIME — Indexes the TIME value, which is not a special character. The use of the TIME value is described in the discussion of non-canonical mode input processing in Understanding the POSIX Line Discipline.

The character values for the following control characters can be changed:

INTR	EOF	STOP	DISCARD
QUIT	EOL	SUSP	WERASE
ERASE	EOL2	DSUSP	LNEXT
KILL	START	REPRINT	

The ERASE, KILL, and EOF characters can also be escaped (preceded with a backslash) so that no special processing is done.

The following values for the *Optional_Actions* parameter of the **tcsetattr** subroutine are also defined in the **termios.h** header file:

TCSANOW — Immediately sets the parameters associated with the terminal from the referenced **termios** structure.

TCSADRAIN — Waits until all output written to the object file has been transmitted before setting the terminal parameters from the **termios** structure.

TCSAFLUSH — Waits until all output written to the object file has been transmitted and all input received but not read has been discarded before setting the terminal parameters from the **termios** structure.

The following values for the *Queue_Selector* parameter of the **tcflush** subroutine are also defined in this header file:

TCIFLUSH — Flushes data that is received but not read.

TCOFLUSH — Flushes data that is written but not transmitted.

TCIOFLUSH — Flushes both data that is received but not read and data that is written but not transmitted.

The following values for the *Action* parameter of the **tcflow** subroutine are also defined in the **termios.h** header file:

TCOOFF	Suspends the output of data by the object file named in the **tcflow** subroutine.
TCOON	Restarts data output that was suspended by the TCOOFF parameter.
TCIOFF	Transmits a stop character to stop data transmission by the terminal device.
TCION	Transmits a start character to start or restart data transmission by the terminal device.

Modem Control Operations

The following **ioctl** operations, which are used for modem control, are an extension to the POSIX line discipline. To use these operations in a program, the program must contain an **#include** statement for the **ioctl.h** header file.

TIOCMBIS	Turns on the control lines specified by the integer mask value of the argument to this command. No other control lines are affected.
TIOCMBIC	Turns off the control lines specified by the integer mask value of the argument to this command. No other control lines are affected. This command operates on the same
TIOCMGET	Gets all modem bits. The argument to this command is a pointer to an integer into which the the current state of the modem status lines is stored.
TIOCMSET	Sets all modem bits. The argument to this command is a pointer to an integer containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.

The integer specifies one of the following modem control or status lines on which the modem control **ioctl** command operates:

TIOCM_LE	Line enable.
TIOCM_DTR	Data terminal ready.
TIOCM_RTS	Request to send.
TIOCM_ST	Secondary transmit.
TIOCM_SR	Secondary receive.
TIOCM_CTS	Clear to send.
TIOCM_CAR	Carrier detect.
TIOCM_CD	TIOCM_CAR.
TIOCM_RNG	Ring.
TIOCM_RI	TIOCM_RNG.
TIOCM_DSR	Data set ready.

These **ioctl** operations are also supported by the **termio** terminal interface.

For general information about the TTY subsystem, refer to TTY Subsystem Overview in *General Programming Concepts*. For general information about the POSIX line discipline, refer to Understanding the POSIX Line Discipline in *General Programming Concepts*.

Files

/usr/include/sys/ioctl.h	The path to the **ioctl.h** header file, which contains the **winsize** structure used by this line discipline. The **termios.h** file contains an **#include** statement for the **ioctl.h** file.
/usr/include/sys/termio.h	The path to the **termio.h** header file, which provides maximum compatibility with the AIX Version 2 terminal interface.
/usr/include/termios.h	The path to the **termios.h** header file.
/usr/include/sys/types.h	The path to the **types.h** header file, which defines primitive system data types.

Implementation Specifics

This file is for POSIX compatibility.

This file is part of AIX Base Operating System (BOS) Runtime.

Related Information

The **termio.h** file, **types.h** file.

The **ioctl** subroutine, **sigvec** subroutine, **tcflow** subroutine, **tcflush** subroutine, **tcsetattr** subroutine.

The **csh** command, **getty** command, **ksh** command, **stty** command, **tset** command.

types.h File

Purpose

Defines primitive system data types.

Description

The data types defined in the **types.h** header file are used in the computer system source code. Some data of these types are accessible to user code and can be used to enhance portability across different machines and operating systems. For example, the pid_t type allows for more processes than the unsigned short type and the dev_t type can be 16 bits rather than 32 bits on a machine that supports fewer devices than the RISC System/6000.

Standard Type Definitions

The **types.h** header file includes the following standard type definitions, which are defined with a typedef statement:

daddr_t	This data type is used for disk addresses, except in i-nodes on disk. The **fs** file format describes the format of disk addresses used in i-nodes.
caddr_t	The core (memory) address.
ino_t	The i-node number (filesystem).
cnt_t	The file system reference count type.
dev_t	The major and minor parts of a device code specify the kind of device and unit number of the device, and they depend on the system customization.
chan_t	The channel number (the minor's minor).
off_t	The file offset, measured in bytes from the beginning of a file.
paddr_t	The real address.
key_t	The ipc key.
timer_t	Timer ID. Times are encoded in seconds, since 00:00:00 GMT, January 1, 1970.
nlink_t	The number of file links.
mode_t	The file mode.
uid_t	The user ID.
gid_t	The group ID.
mid_t	The module ID.
pid_t	The process ID.
slab_t	The security label.
mtyp_t	The ipc message type.

uchar_t	Unsigned char.
ushort_t	Unsigned short.
uint_t	Unsigned int.
ulong_t	Unsigned long.

Unsigned Integers and Addresses

The **types.h** header file also includes the following type definitions for unsigned integers and addresses:

```
typedef struct          _quad { long val[2]; } quad;
typedef long            swblk_t;
typedef long            size_t;
```

The following type definitions are for BSD compatibility only.

```
typedef unsigned char           u_char;
typedef unsigned short  u_short;
typedef unsigned int            u_int;
typedef unsigned long           u_long;
```

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of Includes and Libraries in AIX Base Application Development Toolkit.

File

/usr/include/sys/types.h	The path to the **types.h** header file.

Related Information

The **fs** file format.

The **values.h** header file.

unistd.h File

Purpose

Defines implementation characteristics identified by the IEEE P1003 Portable Operating System Interface for Computer Environments (POSIX).

Description

The **unistd.h** header file includes several other files that contain definitions that are required for compatibility with the IEEE POSIX 1003 standard.

access.h Defines symbolic constants for the **access** subroutine.

lseek.h Defines symbolic constants for the **lseek** subroutine.

The **unistd.h** file also defines symbolic constants for the **pathconf** subroutine, **fpathconf** subroutine, and **sysconf** subroutine. The **unistd.h** header file defines the following symbols, which are used by POSIX 1003 applications to determine implementation characteristics:

_POSIX_JOB_CONTROL POSIX 1003-compatible job control is supported.

_POSIX_SAVED_IDS An **exec** subroutine saves the effective user and group IDs.

_POSIX_VERSION The version of the POSIX standard with which this version of the operating system complies. The value of this symbol is 198808L.

_POSIX_CHOWN_RESTRICTED
The use of the **chown** function is restricted to a process with the appropriate privileges. The group id of a file can be changed only to the effective group id or a supplementary group id of the process. The value of this symbol is –1.

_POSIX_VDISABLE The terminal special characters, which are defined in the **termios.h** header file, can be disabled if this character value is defined by the **tcsetattr** subroutine. The value of this symbol is –1.

_POSIX_NO_TRUNC Path name components that are longer than NAME_MAX will generate an error.

The **unistd.h** file also defines the following symbol, which is used by X/OPEN applications:

_XOPEN_VERSION The version of the X/OPEN standard with which this version of the operating system complies.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is provided for POSIX compatibility.

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/unistd.h The path to the **unistd.h** header file.

Related Information

The **access** subroutine, **exec** subroutine.

The **values.h** header file.

utmp.h File

Purpose

Defines the structures of certain user and accounting information files.

Description

The structure of the records in the **utmp**, **wtmp**, and **failedlogin** files is defined in the **utmp.h** header file. The **utmp** structure in this header file contains the following fields:

ut_user — The user login name.

ut_line — The device name (console or ln*xx*). The maximum length of a string in this field is 11 characters plus a null character. When accounting for something other than a process, the following special strings or formats are allowed:

RUNLVL_MSG	Run level. The run level of the process.
BOOT_MSG	System boot. The time of the initial program load (IPL).
OTIME_MSG	Old time. The time of login.
NTIME_MSG	New time. The time idle.

ut_pid — The process id.

ut_type — The type of entry, which can be one of the following values:

EMPTY	Unused space in file.
RUN_LVL	The run level of the process, as defined in the **inittab** file.
BOOT_TIME	The time at which the system was started.
OLD_TIME	The time at which a user logged on to the system.
NEW_TIME	The amount of time the user is idle.
INIT_PROCESS	A process spawned by the **init** command.
LOGIN_PROCESS	A **getty** process waiting for a login.
USER_PROCESS	A user process.
DEAD_PROCESS	A zombie process.
ACCOUNTING	A system accounting process.
UTMAXTYPE ACCOUNTING	The largest legal value allowed in the ut_type field.

Embedded within the **utmp** structure is the **exit_status** structure, which contains the following fields:

e_termination — The termination status of a process.

e_exit — The exit status of a process, marked as the DEAD_PROCESS value.

ut_time — The time at which the entry was made.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of Accounting Services in AIX BOS Extensions 2.

Files

/etc/utmp	The path to the **utmp** file, which contains a record of users logged into the system.
/usr/adm/wtmp	The path to the **wtmp** file, which contains accounting information about users logged in.
/etc/.ilog	The path to the **failedlogin** file, which contains a list of invalid login attempts.
/usr/include/utmp.h	The path to the **utmp.h** header file.

Related Information

The **getty** command, **init** command, **login** command, **who** command, **write** command.

The **utmp**, **wtmp**, and **failedlogin** files.

values.h File

Purpose

Defines machine-dependent values.

Description

The **values.h** header file contains a set of 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 the HIBITL value).

MAXSHORT The maximum value of a signed short integer (0x7FFF = 32,767).

MAXLONG The maximum value of a signed long integer (0x7FFFFFFF = 2,147,483,647).

MAXINT The maximum value of a signed regular integer (the same as the MAXLONG value).

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 the MAXDOUBLE value.

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 maximum exponent of a single-precision floating-point number.

DMAXEXP The maximum 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.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/values.h The path to the **values.h** header file.

Related Information

The **math.h** file, **types.h** file.

vmount.h File

Purpose

Defines the structure of the data associated with a virtual file system.

Description

The **vmount.h** header file defines the **vmount** structure. Each active virtual file system has a **vmount** structure associated with it. The **vmount** structure contains the mount parameters (such a the mount object and the mounted over object) for that virtual file system. The **vmount** data is created when the virtual file system is mounted; the **mntctl** subroutine returns the virtual file system data.

The **vmount** structure contains the following fields to describe fixed-length data:

vmt_revision — The revision code in effect when the program that created this virtual file system was compiled.

vmt_length — The total length of the structure and data. This will always be a multiple of the word size (4 bytes).

vmt_fsid — The two-word file system identifier; the interpretation of this identifier depends on the vmt_gfstype field.

vmt_vfsnumber — The unique identifier of the virtual file system. Virtual file systems and their identifiers are deleted at IPL (initial program load).

vmt_time — The time at which the virtual file system was created.

vmt_flags — The general mount flags, for example: READONLY, REMOVABLE, DEVICE, REMOTE.

vmt_gfstype — The type of the general file system. Possible values are:

MNT_JFS	The AIX Version 3 journalled file system.
MNT_NFS	The SUN network file system.
MNT_CDROM	CD-ROM file system.

The remaining fields in the **vmount** structure describe variable-length data. Each entry in the vmt_data array specifies the offset from the start of the **vmount** structure at which a data item appears as well as the length of the data item.

vmt_off — The offset of the data, aligned on a word (32-bit) boundary.

vmt_size — The actual size of the data, in bytes.

vmt_data[VMT_OBJECT]
The name of the device, directory, or file that is mounted.

vmt_data[VMT_STUB]
The name of the device, directory, or file that is mounted over.

vmt_data[VMT_HOST]
The short (binary) name of the host that owns the mounted object.

vmt_data[VMT_HOSTNAME]
The long (character) name of the host that owns the mounted object.

vmt_data[VMT_INFO]
: Binary information passed to the file system implementation that supports this object; the contents of this field is specific to the gfs type defined by the vmt_gfstype field.

vmt_data[VMT_ARGS]
: A character string representation of the arguments supplied when the virtual file system was created.

Also refer to the Header Files Overview, which defines header files, describes how they are used, and lists several of the AIX header files for which information is provided in this documentation.

Implementation Specifics

This file is part of AIX Base Operating System (BOS) Runtime.

File

/usr/include/sys/vmount.h The path to the **vmount.h** header file.

Related Information

The **mntctl** subroutine, **uvmount** subroutine, **vmount** subroutine.

HCON fxconst.inc File

Purpose

Provides **fxfer** function constants for Pascal version file transfer.

Description

The **/usr/include/fxconst.inc** file contains the constants used in programmatic Pascal file transfer program. Each module that uses the Pascal file transfer function must include the **fxconst.inc** file. The constants are for use with Pascal program interface to the HCON File Transfer Program.

The following constants are for the **f_flags** variable:

```
FXC_UP          = 1;            /* '0001'x */
FXC_DOWN        = 2;            /* '0002'x */
FXC_TNL         = 4;            /* '0004'x */
FXC_TCRLF       = 8;            /* '0008'x */
FXC_REPL        = 16;           /* '0010'x */
FXC_APPND       = 32;           /* '0020'x */
FXC_QUEUE       = 64;           /* '0040'x */
FXC_FIXED       = 128;          /* '0080'x */
FXC_VAR         = 256;          /* '0100'x */
FXC_UNDEF       = 512;          /* '0200'x */
FXC_TSO         = 1024;         /* '0400'x */
FXC_CMS         = 2048;         /* '0800'x */
```

The following constants are for the allocation variables:

```
FXC_TRACKS          = -1;        /* Tracks   */
FXC_CYLINDERS       = -2;        /* Cylinder */
```

Implementation Specifics

The **fxconst.inc** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a Pascal compiler.

Related Information

File transfer functions are the **g32_fxfer** function, **cfxfer** function, **fxfer** function.

HCON fxfer.h File

Purpose

Contains the **fxc** and **fxs** data structures for the C file transfer functions.

Description

The **/usr/include/fxfer.h** file contains the structures used in the C file transfer program. Each module that uses the C file transfer function must include the **fxfer.h** file. The structures are for use with C program interface to the HCON File Transfer Program.

The **fxc** structure fields are as follows:

```
struct fxc {
   char *fxc_src;            /* Source file name                */
   int srclength;            /* Put here for Pascal stringptr   */
   char *fxc_dst;            /* Destination file name           */
   int dstlength;            /* Put here for Pascal stringptr   */
   struct fxcf {
      int f_flags;
#define FXC_UP    0x0001
#define FXC_DOWN  0x0002
#define FXC_TNL   0x0004
#define FXC_TCRLF 0x0008
#define FXC_REPL  0x0010
#define FXC_APPND 0x0020
#define FXC_QUEUE 0x0040
#define FXC_FIXED 0x0080
#define FXC_VAR   0x0100
#define FXC_UNDEF 0x0200
#define FXC_TSO   0x0400
#define FXC_CMS   0x0800
      char *f_logonid;        /* Logon id                        */
      int  loglength;         /* Put here for Pascal stringptr */
      int  f_lrecl;           /* Logical record length         */
      int  f_blksize;         /* Block size                    */
      struct fxcs {
         int s_space;         /* Allocation space              */
         int s_increment;     /* Allocation space increment    */
         int s_unit;          /* Unit of allocation            */
#define FXC_TRACKS   -1       /* Tracks                        */
#define FXC_CYLINDERS -2      /* Cylinder                      */
      } f_s;
   } fxc_opts;
};

struct fxs {
   int  fxs_bytcnt;           /* Byte count                     */
   char *fxs_src;             /* Source file name               */
   int  srclen;               /* Put here for Pascal stringptr  */
   char *fxs_dst;             /* Destination file name          */
   int  dstlen;               /* Put here for Pascal stringptr  */
   char *fxs_ctime;           /* Destination file creation time */
   int  timelen;              /* Put here for Pascal stringptr  */
   int  fxs_stat;             /* Status code                    */
   int  fxs_errno;            /* Errno                          */
};
```

```
struct fxp {
   char *prof_id;              /* Profile id                      */
   int  proflen;               /* Put here for Pascal stringptr   */
};
```

Implementation Specifics

The **fxfer.h** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a C compiler.

Related Information

File transfer functions are the **g32_fxfer** function, **cfxfer** function, **fxfer** function.

HCON fxfer.inc File

Purpose

Contains the **fxc** and **fxs** records for Pascal file transfer functions.

Description

The **/usr/include/fxfer.inc** file contains the **fxc** and **fxs** records used in the programmatic pascal file transfer program. Each module that uses the Pascal file transfer function must include the **fxfer.inc** file. The records are for use with Pascal program interface to the HCON Programmatic File Transfer.

The **fxs** record fields are as follows:

```
fxs = record
   fxs_bytcnt : integer;      /* Byte count                        */
   fxs_src    : stringptr;    /* Source file name                 */
   fxs_dst    : stringptr;    /* Destination file name            */
   fxs_ctime  : stringptr;    /* Destination file creation time */
   fxs_stat   : integer;      /* Status code                      */
   fxs_errno  : integer;      /* Errno                            */
end;                          /* Record fxs                       */
```

The **fx_s** record fields are as follows:

```
fx_s = record
   s_space     : integer;     /* Allocation space                */
   s_increment : integer;     /* Alloction space increment       */
   s_unit      : integer;     /* Unit of allocation              */
end;                          /* Record f_s                      */

fxc_opt = record              /* Options record                  */
   f_flags    : integer;      /* Flags options                   */
   f_logonid : stringptr;    /* Address of logon id string    */
   f_lrecl   : integer;      /* Logical record length         */
   f_blksize : integer;      /* Block size                    */
   f_s       : fx_s;          /* S option record                 */
end;                          /* Record fxc_opts                 */
```

The **fxc** record fields are as follows:

```
fxc = record
   fxc_src  : stringptr;      /* Source file name                */
   fxc_dst  : stringptr;      /* Destination file name           */
   fxc_opts : fxc_opt;        /* Options record                  */
end;                          /* Record fxc                      */
```

Implementation Specifics

The **fxfer.inc** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a Pascal compiler.

Related Information

The **g32_fxfer** function, **fxfer** function, **cfxfer** function.

HCON fxhfile.inc File

Purpose

Contains external declarations for Pascal file transfer.

Description

The **/usr/include/fxhfile.inc** header file provides external definitions for the **pfxfer** File Transfer Pascal Program status file. The **fxhfile.inc** is the Pascal file transfer invocation file. Each module that uses the Pascal file transfer function must include the **fxhfile.inc** file. The fields in the **fxhfile.inc** file are:

```
function pfxfer(var xfer : fxc; comm : stringptr):integer;external;

function pcfxfer(var sfer : fxs):integer;external;
```

Implementation Specifics

The **fxhfile.inc** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a Pascal compiler.

Related Information

The **g32_fxfer** function, **fxfer** function, **cfxfer** function.

HCON g32_api.h File

Purpose

Contains associated API symbol definitions and data structures.

Description

The **/usr/include/g32_api.h** file provides data definitions and structures for use with HCON C language subroutines. Each module that uses the HCON API must include the **g32_api.h** file.

The constants in the **g32_api** file are:

```
#define H3270DEV            0
#define SS1                 0x19
/*
 *      Range for logical path ID's.
 */
#define MIN_LPID            0
#define MAX_LPID            15
#define NUM_LPS             16

#define G32OK               0
#define G32ERROR            -1

#define NO_SESSION          0
#define MODE_3270           1
#define MODE_API            2
#define MODE_API_T          4
#define PEND_DEALLOC        8

#define MAX_MSG_LEN         60000
/*
 * screen size definitions for the different models
 */
#define MIN_ROW             1
#define MIN_COLUMN          1
#define MOD2_MAXROW         24
#define MOD2_MAXCOL         80
#define MOD3_MAXROW         32
#define MOD3_MAXCOL         80
#define MOD4_MAXROW         43
#define MOD4_MAXCOL         80
#define MOD5_MAXROW         27
#define MOD5_MAXCOL         132
```

g32_api structure

```
struct g32_api {     /* information and parameter structure  */
    int lpid;        /* logical path id                      */
    int errcode;     /* error code indicator                 */
    int xerrinfo;    /* extra error information              */
    int row;         /* row number                           */
    int column;      /* column number                        */
    int length;      /* length for patterns                  */
    int eventf;      /* message queue ID/file descriptor     */
    int maxbuf;      /* maximum buffer size                  */
    int timeout;     /* timeout of host response             */
};

/*
  * This structure was put in to have a structure that directly
  * corresponded to a Pascal stringptr (which equals a char
   *  and int).
 */
struct g32_str {
    char *g_strval;
    int  g_strlength;
};

extern int errno;
/*
 *      Error codes used by the API routines
 */
#define G32_SESS_EXIST   -1  /* A session exists on the logical
                                path*/
#define G32_NO_LA        -2 /*  There are no free link addresses */
#define G32_EOPEN        -3  /* An error occurred opening the
                                special file */
#define G32_NO_LOG       -5  /* An error occurred while attempting
                                log onto the host */
#define G32_NO_LP        -6  /* No logical path was available */
#define G32_NO_SESS      -7  /* No sesssion exists for this
                                application  */

#define G32_EEMU         -8  /* Error starting emulator */
#define G32_EMALLOC      -9  /* Unable to malloc memory */
#define G32_EFORK        -10 /* fork failed */
#define G32_ENDSESS      -12 /* The host application wishes to end
                                the session */
#define G32_INV_MODE     -13 /* The AIX application is not in
                                API/API or API/API_T mode */
#define G32_MIX_MODE     -14 /* The session is in API/3270 versus
                                host API/API application  */

#define G32_PARMERR      -15 /* No host application name was
                                specified for an API or API_T mode
                                application  */
#define G32_LINK_CTL     -16    /* The api was unable to get control
                                of the specified logical path */
#define G32_EREAD        -17    /* An error occurred on a 'read'  */
#define G32_EWRITE       -18    /* An error occurred on a 'write'*/
#define G32_ELENGTH      -19    /* The message is more than 32000
                                bytes long, or negative */
#define G32_INV_POSITION -20    /* The row or column specification
                                was invalid */
```

```
#define G32_INV_PATTERN  -21    /* The pattern presented to a
                                   G32_search was invalid */
#define G32_SEARCH_FAIL  -23    /* The string was not found in the
                                   presentation space */
#define G32_EMSGSND      -24    /* The API was not able to send a
                                   msg to the emulator */
#define G32_EMSGRCV      -25    /* The API was not able to receive a
                                   msg from the emulator */
#define G32_PROMPT       -29    /* The api was not able to read the
                                   control terminal for a logon ID and
                                   passwd */
#define G32_EIOCTL       -30    /* The ioctl call to the AIX driver
                                   failed  */
#define G32_ESELECT      -31    /* An error occurred on a select */
#define G32_NOTACK       -32     /* The synchronization problem,
                                   missing g32write function in the
                                   host application */
#define G32_TIMEOUT      -33    /* A timeout occurred waiting for
                                   host data */

/*
 *      Codes returned by g32_get_status
 */
#define G32_NO_ERROR    0
#define G32_COMM_CHK    -1
#define G32_PROG_CHK    -2
#define G32_MACH_CHK    -3

/*
 *      constants used in g32_openx
 */
#define ASCII_0         060
#define ASCII_1         061
#define ASCII_9         071

/*
 *      length of header
 */
#define HEADER_LENGTH   12


/*
 *      values for emulator quit message
 */
#define QUIT_BYTE1      0x03
#define QUIT_BYTE2      0x01
#define QUIT_BYTE3      0x00

/*
 *      values used in g_sea_xlate
 */
#define HEXa0           0xa0
#define HEXb4           0xb4
#define HEXb5           0xb5
#define HEXc0           0xc0
#define HEXe6           0xe6
```

```
/*
 *      values used in g32_alloc and g32_write
 */
#define MAX_BUF_DIV_256 7
#define MAX_BUF_MOD_256 8

/*
 *      value used in g32_alloc
 */
#define STRUCT_AID      0x88
```

Implementation Specifics

The **g32_api.h** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a C compiler.

Related Information

HCON AIX Header Files and Understanding the AIX Interface in *Communications Programming Concepts.*

AIX interface functions are:

g32_alloc	**g32_get_cursor**	**g32_get_status**
g32_close	**g32_get_data**	**g32_read**
g32_dealloc	**g32_search**	**g32_write**
g32_openx	**g32_send_keys**	**g32_fxfer**

HCON g32const.inc File

Purpose

Defines Pascal HCON API constants

Description

The **/usr/include/g32const.inc** file contains definitions for API constants to use with HCON Pascal language subroutines. Each module that uses the Pascal API must include the **g32const.inc** file.

The constants in the **g32const.inc** file are:

```
        H3270DEV        = 0;
        SS1             = '19'x;
/*
 *      Range for logical path IDs.
 */
        MIN_LPID        = 0;
        MAX_LPID        = 15;
        NUM_LPS = 16;

        G32OK           = 0;
        G32ERROR        = -1;

        NO_SESSION      = 0;
        MODE_3270       = 1;
        MODE_API        = 2;
        MODE_API_T      = 4;
        PEND_DEALLOC    = 8;

        MAX_MSG_LEN     = 60000;

        API_USER_MSG    = '01'x;
        API_START_MSG   = '02'x;
        API_TERM_MSG    = '03'x;
        WSF             = '11'x;
        API_SMSG_LEN    = 11;
        API_TMSG_LEN    = 11;
        API_NMSG_LEN    = 11;
        API_HDR_LEN     = 11;


/*
 *      Error codes used by the API routines
 */
        G32_SESS_EXIST  = -1;
        G32_NO_LA               = -2;
        G32_EOPEN               = -3;
        G32_NO_LOGON            = -5;
        G32_NO_LP               = -6;
        G32_NO_SESS             = -7;
        G32_EEMU                = -8;
        G32_EMALLOC             = -9;
        G32_EFORK               = -10;
        G32_ENDSESS             = -12;
        G32_INV_MODE            = -13;
        G32_PARMERR             = -15;
```

```
        G32_LINK_CTL            = -16;
        G32_EREAD               = -17;
        G32_EWRITE              = -18;
        G32_ELENGTH             = -19;
        G32_INV_POSITION         = -20;
        G32_INV_PATTERN = -21;
        G32_SEARCH_FAIL = -23;
        G32_EMSGSND             = -24;
        G32_EMSGRCV             = -25;
        G32_PROMPT              = -29;
        G32_EIOCTL              = -30;
        G32_ESELECT             = -31;
        G32_NOTACK              = -32;
        G32_TIMEOUT             = -33;

/*
 *      Codes returned by g32stat
 */
        G32_NO_ERROR    = 0;
        G32_COMM_CHK    = -1;
        G32_PROG_CHK    = -2;
        G32_MACH_CHK    = -3;
```

Implementation Specifics

The **g32const.inc** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a Pascal compiler.

Related Information

HCON AIX Header Files and Understanding the AIX Interface in *Communications Programming Concepts*.

AIX interface functions are:

g32_alloc	**g32_get_cursor**	**g32_get_status**
g32_close	**g32_get_data**	**g32_read**
g32_dealloc	**g32_search**	**g32_write**
g32_openx	**g32_send_keys**	**g32_fxfer**

HCON g32hfile.inc File

Purpose

Contains HCON API external definitions for Pascal language.

Description

The **/usr/include/g32hfile.inc** file provides external definitions for use with HCON Pascal language subroutines. Each module that uses the Pascal API must include the **g32hfile.inc** file.

The function declarations in the **g32hfile.inc** file are:

```
function g32allc(var as : g32_api;
          appl_name : stringptr;
          session_mode : integer):integer;external;

function g32clse(var as : g32_api ):integer;external;

function g32curs(var as : g32_api):integer;external;

function g32deal(var as : g32_api):integer;external;

function g32data(var as : g32_api;
          buffer : integer):integer;external;

function g32fxfer(var as : g32_api;
          xfer : fxc):integer;external;

function g32note(var as : g32_api;
          note : integer):integer;external;

function g32open(var as : g32_api;
          flag : integer;
          uid : stringptr;
          pw : stringptr;
          comm : stringptr):integer;external;

function g32openx(var as : g32_api;
          flag : integer;
          uid : stringptr;
          pw : stringptr;
          comm : stringptr;
          timeout : stringptr):integer;external;

function g32read(var as : g32_api;
          var buffer : stringptr;
          var msg_len : integer):integer;external;

function g32sdky(var as : g32_api;
          buffer : stringptr):integer;external;

function g32srch(var as : g32_api;
          pattern : stringptr):integer;external;

function g32stat(var as : g32_api):integer;external;

function g32wrte(var as : g32_api;
          buffer : integer;
          msg_len : integer):integer;external;
```

Implementation Specifics

The **g32hfile.inc** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a Pascal compiler.

Related Information

HCON AIX Header Files and Understanding the AIX Interface in *Communications Programming Concepts.*

AIX interface functions are:

g32_alloc	**g32_get_cursor**	**g32_get_status**
g32_close	**g32_get_data**	**g32_read**
g32_dealloc	**g32_search**	**g32_write**
g32_openx	**g32_send_keys**	**g32_fxfer**

HCON g32_keys.h File

Purpose

Enables HCON API in Mode_3270 for C subroutines.

Description

The **/usr/include/g32_keys.h** file enables HCON API in Mode_3270 for use with the HCON C language **g32_send_keys** function. Each module that uses the HCON Pascal **g32_send_keys** function must include **g32_keys.h** file.

The constants in the **g32_keys.h** file are:

```
#define ENTER    "\002\061"      /* enter  */
#define PA1      "\002\055"      /* PA1    */
#define PA2      "\002\056"      /* PA2    */
#define PA3      "\002\057"      /* PA3     */
#define PF1      "\002\025"      /* PF1     */
#define PF2      "\002\026"      /* PF2     */
#define PF3      "\002\027"      /* PF3     */

#define PF4      "\002\030"      /* PF4     */
#define PF5      "\002\031"      /* PF5     */
#define PF6      "\002\032"      /* PF6     */
#define PF7      "\002\033"      /* PF7     */
#define PF8      "\002\034"      /* PF8     */
#define PF9      "\002\035"      /* PF9     */
#define PF10     "\002\036"      /* PF10    */

#define PF11     "\002\037"      /* PF11    */
#define PF12     "\002\040"      /* PF12    */
#define PF13     "\002\041"      /* PF13    */
#define PF14     "\002\042"      /* PF14    */
#define PF15     "\002\043"      /* PF15    */
#define PF16     "\002\044"      /* PF16    */
#define PF17     "\002\045"      /* PF17    */
#define PF18     "\002\046"      /* PF18    */
#define PF19     "\002\047"      /* PF19    */

#define PF20     "\002\050"      /* PF20    */
#define PF21     "\002\051"      /* PF21    */
#define PF22     "\002\052"      /* PF22    */
#define PF23     "\002\053"      /* PF23    */
#define PF24     "\002\054"      /* PF24    */
#define CLEAR    "\002\060"      /* clear   */
#define DUP      "\002\066"      /* dup     */

#define FM       "\002\067"      /* field mark      */
#define INS      "\002\024"      /* insert          */
#define DEL      "\002\021"      /* delete          */
#define C_UP     "\002\002"      /* cursor up       */
#define C_DN     "\002\003"      /* cursor down     */
#define C_LT     "\002\001"      /* cursor left     */
#define C_RT     "\002\004"      /* cursor right     */
#define C_UUP    "\002\006"      /* cursor up fast   */
#define C_DDN    "\002\007"      /* cursor down fast */
#define C_LLT    "\002\005"      /* cursor left fast */
```

```
#define C_RRT    "\002\010"      /* cursor right fast     */
#define TAB      "\002\013"      /* tab                   */
#define B_TAB    "\002\014"      /* back tab              */
#define CR       "\002\012"      /* carriage return       */
#define RESET    "\003\002"      /* reset                 */
#define E_INP    "\002\022"      /* erase input           */
#define E_EOF    "\002\023"      /* erase to end of field */
#define T_REQ    "\002\070"      /* test/sys req          */
#define HOME     "\002\015"      /* home cursor           */
```

Implementation Specifics

The **g32_keys.h** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a C compiler.

Related Information

The **g32_send_keys** function

HCON AIX Header Files and Understanding the AIX Interface in *Communications Programming Concepts.*

HCON g32keys.inc File

Purpose

Contains common API key values definitions.

Description

The **/usr/include/g32keys.inc** file provides key definitions for use with the HCON Pascal language **g32_send_keys** function. Each module that uses the HCON Pascal **g32_send_keys** function must include **g32keys.inc**.

The key value definitions in the **g32keys.inc** file are:

```
ENTER   = chr(2) || chr(49);    /* enter key (host) */
PA1     = chr(2) || chr(45);    /* PA1  */
PA2     = chr(2) || chr(46);    /* PA2  */
PA3     = chr(2) || chr(47);    /* PA3  */
PF1     = chr(2) || chr(21);    /* PF1  */
PF2     = chr(2) || chr(22);    /* PF2  */

PF3     = chr(2) || chr(23);    /* PF3  */
PF4     = chr(2) || chr(24);    /* PF4  */
PF5     = chr(2) || chr(25);    /* PF5  */
PF6     = chr(2) || chr(26);    /* PF6  */
PF7     = chr(2) || chr(27);    /* PF7  */

PF8     = chr(2) || chr(28);    /* PF8  */
PF9     = chr(2) || chr(29);    /* PF9  */
PF10    = chr(2) || chr(30);    /* PF10  */
PF11    = chr(2) || chr(31);    /* PF11  */
PF12    = chr(2) || chr(32);    /* PF12  */
PF13    = chr(2) || chr(33);    /* PF13  */

PF14    = chr(2) || chr(34);    /* PF14  */
PF15    = chr(2) || chr(35);    /* PF15  */
PF16    = chr(2) || chr(36);    /* PF16  */
PF17    = chr(2) || chr(37);    /* PF17  */
PF18    = chr(2) || chr(38);    /* PF18  */

PF19    = chr(2) || chr(39);    /* PF19  */
PF20    = chr(2) || chr(40);    /* PF20  */
PF21    = chr(2) || chr(41);    /* PF21  */
PF22    = chr(2) || chr(42);    /* PF22  */
PF23    = chr(2) || chr(43);    /* PF23  */
PF24    = chr(2) || chr(44);    /* PF24  */

CLEAR   = chr(2) || chr(48);    /* clear      */
DUP     = chr(2) || chr(54);    /* dup        */
FM      = chr(2) || chr(55);    /* field mark */
INS     = chr(2) || chr(20);    /* insert     */
DEL     = chr(2) || chr(17);    /* delete     */

C_UP    = chr(2) || chr(2);     /* cursor up     */
C_DN    = chr(2) || chr(3);     /* cursor down     */
C_LT    = chr(2) || chr(1);     /* cursor left     */
C_RT    = chr(2) || chr(4);     /* cursor right    */
C_UUP   = chr(2) || chr(6);     /* cursor up fast  */
```

```
C_DDN    = chr(2) || chr(7);     /* cursor right fast   */
C_LLT    = chr(2) || chr(5);     /* cursor left fast    */
C_RRT    = chr(2) || chr(8);     /* cursor right fast   */
TAB      = chr(2) || chr(11);    /* tab               */
B_TAB    = chr(2) || chr(12);    /* back tab            */

CR       = chr(2) || chr(10);    /* carriage return       */
RESET    = chr(3) || chr(2);     /* reset                 */
E_INP    = chr(2) || chr(18);    /* erase input           */
E_EOF    = chr(2) || chr(19);    /* erase to end of field */
T_REQ    = chr(2) || chr(56);    /* test/sys request      */
HOME     = chr(2) || chr(13);    /* home cursor           */
```

Implementation Specifics

The **g32_keys.inc** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a Pascal compiler.

Related Information

The **g32_send_keys** function

HCON AIX Header Files and Understanding the AIX Interface in *Communications Programming Concepts*.

HCON g32types.inc File

Purpose

Contains Pascal API data types.

Description

The **/usr/include/g32types.inc** file provides data types and structures for use with HCON Pascal language functions. The **g32types.inc** file is an include file that contains the **g32_api** record. Each module that uses the HCON Pascal API must include **g32types.inc**.

The fields in the **g32types.inc** file are:

```
g32_api = record          /* information and parameter structure  */
    lpid : integer;        /* logical path id                      */
    errcode : integer;     /* error code indicator                 */
    xerrinfo : integer;    /* extra error information              */
    row : integer;         /* row number                           */
    column : integer;      /* column number                        */
    length : integer;      /* length for patterns                  */
    eventf : integer;      /* message queue ID/file descriptor     */
    maxbuf : integer;      /* the maximum transfer message size    */
                           /* from the H3270MAXBUF variable in     */
                           /* the HCON profile.                    */
                           /* The user may override the default    */
                           /* value only during a call to          */
                           /* g32allc                              */
    timeout : integer;     /* the amount of time, in seconds,      */
                           /* to wait for data from the host       */
                           /* computer.  The default value is      */
                           /* 15 seconds.  The user may override   */
                           /* the default value at anytime.        */

end; /* record g32_api */

fxs = record
   fxs_bytcnt : integer;       /* Byte count                        */
   fxs_src    : stringptr;     /* Source file name                  */
   fxs_dst    : stringptr;     /* Destination file name             */
   fxs_ctime  : stringptr;     /* Destination file creation time    */
   fxs_stat   : integer;       /* Status code                       */
   fxs_errno  : integer;       /* Errno                             */
end;                           /* Record fxs                        */

fx_s = record
   s_space     : integer;      /* Allocation space                  */
   s_increment : integer;      /* Alloction space increment         */
   s_unit      : integer;      /* Unit of allocation                */
end;                           /* Record f_s                        */

fxc_opt = record               /* Options record                    */
   f_flags   : integer;        /* Flags options                     */
   f_logonid : stringptr;      /* Address of logon id string        */
   f_lrecl   : integer;        /* Logical record length             */
   f_blksize : integer;        /* Block size                        */
   f_s       : fx_s;           /* S option record                   */
end;                           /* Record fxc_opts                   */
```

```
fxc = record
   fxc_src  : stringptr;      /* Source file name              */
   fxc_dst  : stringptr;      /* Destination file name         */
   fxc_opts : fxc_opt;        /* Options record                */
end;
```

Implementation Specifics

The **g32types.inc** file is part of the AIX 3270 Host Connection Program/6000 (HCON).

This file requires the use of a Pascal compiler.

Related Information

HCON AIX Header Files and Understanding the AIX Interface in *Communications Programming Concepts.*

AIX interface functions are:

g32_alloc	**g32_get_cursor**	**g32_get_status**
g32_close	**g32_get_data**	**g32_read**
g32_dealloc	**g32_search**	**g32_write**
g32_openx	**g32_send_keys**	**g32_fxfer**

SNA luxsna.h File

Purpose

Defines constants and structures used by AIX SNA Services/6000 subroutines.

Description

The **luxsna.h** header file provides definitions of constants and structures that are used by AIX SNA Services/6000 subroutines. The **luxsna.h** file defines the following structures:

Structures

The **luxsna.h** file contains the following structure definitions:

allo_str Structure

This structure provides additional parameters for the **snalloc** subroutine and the **ioctl**(ALLOCATE) subroutine. Refer to the **snalloc** or **ioctl** subroutine for a description of these functions. The structure appears as follows:

```
struct allo_str
{
    char        mode_name[9];
    char        tpn[65];
    unsigned    priority                : 2;
    unsigned    type                    : 2;
    unsigned    return_control          : 2;
    unsigned    sync_level              : 2;
    unsigned    recov_level             : 2;
    unsigned    pip                     : 1;
    unsigned    sess_type               : 1;
    unsigned    svc_tpn_flag            : 1;
    unsigned    rsvd2                   : 3;
    int         sense_code;
    long        rid;          /* rid for a previous conversation */
    struct      pip_str  *pip_ptr;
    int         conv_group_id;
} ;
```

Note: When parameters in this structure are specified, the remote TPN profile parameters are overriden. If no parameters are specified, the remote TPN profile parameters are used as the default.

The **allo_str** structure parameters have the following meaning:

`mode_name` Specifies the mode name for the conversation. The mode name designates the network properties for the session to be allocated, such as the class of service to be used.

A special mode name, SNASVCMG, specifies the mode name used for control operator subroutines. Any program using this mode name must have control operator privileges based on the operating-system group ID. This mode name is not used for LUs 1, 2, or 3.

`tpn` Specifies the name of the remote transaction program with which to establish the conversation. The TPN must be coded in EBCDIC. The AIX SNA Services/6000 subroutine interface converts ASCII to EBCDIC if the svc_tpn_flag is set to zero. If you use the AIX SNA Services/6000 subroutine interface, the svc_tpn_flag is ignored, and you must ensure that the conversion is done. Refer to EBCDIC to ASCII Translation for US English (TEXT) for assistance in converting ASCII to EBCDIC.

`priority` Specifies the priority option which selects a mode profile to be used to establish an appropriate session for the conversation. Priority options are:

B'00' Use the first mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.

B'01' Use the second mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.

B'10' Use the third mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.

B'11' Use the fourth mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.

`type` Specifies the type of conversation to be allocated:

DEF_CONV (B'00')
Use the value defined in the remote transaction profile.

BASIC_CONV (B'01')
Allocate a basic conversation.

MAPPED_CONV (B'10')
Allocate a mapped conversation. Do not use this value for LUs 1, 2, or 3.

RECON_CONV (B'11')
Reconnect a conversation between the local and remote transaction program. Do not use this value with LU 1, 2, or 3 conversations.

`return_control`
Specifies the type of conversation the application is requesting. Options are:

WHEN_SESSION_ALLOC (B'00')
Return control to application when either a Contention Winner or Contention Loser session is allocated. The type assigned is the first available.

WHEN_CONWINNER_ALLOCATED (B'01')
Return control to application when a Contention Winner session is allocated.

WHEN_CGID_ALLOCATED (B'10')
Return control to application when a session with the specified conversation group id is allocated.

RESERVED (B'11')
Reserved.

`sync_level` Specifies the synchronization level to be used by the program for this conversation.

DEFAULT (B'00')
Use the value defined in the remote transaction profile.

SYNC_NONE (B'01')
The program does not perform confirmation.

SYNC_CONF (B'10')
The program performs confirmation processing.

`recov_level` Specifies the recovery level that the local program uses for this conversation. Do not use this parameter with LUs 1, 2, and 3.

RECOV_DEF (B'00')
Use the value defined in the remote transaction profile.

RECOV_NONE (B'01')
The program does not support program reconnect or sync point restart.

RECOV_RECON (B'10')
The program supports program reconnect, but not sync point restart.

`PIP` Specifies whether program initialization data is provided for the remote transaction program. Do not use this parameter with LU 1, 2, or 3 conversations.

PIP_NO (B'0')
Program initialization data is not provided.

PIP_YES (B'1')
Program initialization data is provided.

`sess_type` Specifies the type of session to be allocated. Use for LU 1, 2, or 3 sessions only. This field is not used for mapped conversations.

B'0' LU–LU session

B'1' SSCP–LU session

`svc_tpn_flag`
Indicates whether the `tpn` parameter defines a service transaction program name specified in hex:

B'0' Indicates that it is not a service transaction program name. The AIX SNA Services/6000 subroutine interface translates the TPN name into EBCDIC code.

B'1' Indicates that it is a service transaction program name. AIX SNA Services/6000 does not translate the TPN name into EBCDIC code. Refer to EBCDIC to ASCII Translation for US English (TEXT) for assistance in converting ASCII to EBCDIC.

`rsvd2` This field is not used.

`rid` Specifies a resource ID that is returned from an **ioctl**(ALLOCATE) or a **snalloc** subroutine. This field is also used to supply the rid necessary in order to reconnect to an old conversation when the type parameter of the **allo_str** structure is specified as *reconnect* (B'11'). For the remote attached ALLOCATE, the only parameter required in the **allo_str** structure is `rid`, which is passed into the application program by argv [3].

`pip_ptr` When the PIP parameter indicates that program initialization data for the remote program is being supplied, this pointer points to the structure that contains that initialization data. Refer to the **pip_str** structure for **PIP** data structure.

`conv_group_id` The conversation group ID that identifies a specific session to be allocated.

alloc_listen Structure

This structure provides additional parameters for the **ioctl**(ALLOCATE_LISTEN) subroutine. Refer to the **ioctl** subroutine for a description of this function.

```
struct alloc_listen
{
        int                     tpn_mask
        char                    tpn_profile_name[ MAX_PROF_LEN ];
        unsigned short  num_tpn;
        char                    tpn_list[ MAX_NUM_TPN ][MAX_TPN_LEN
];
} ;
```

The **alloc_listen** structure parameters have the following meaning:

`tpn_profile_name`
Specifies the name of a Transaction Program Profile against which incoming attaches are checked. This profile serves as conversation characteristics template for the Transaction Programs listed in the `tpn_list` field. The name can be up to 15 bytes long and *must* be **NULL** terminated (total of 16 bytes maximum). This profile name must be defined in the TPN List Profile for the specified connection.

`num_tpn` The number of TPNs in the `tpn_list` array described below.

`tpn_list` An array of Transaction Program Names to register as being "listened" for. A maximum of 31 names can be registered per call. Each name can be a maximum of 64 bytes and *must* be **NULL** terminated (total of 65 bytes per name). These TPNs should not be defined in the TPN List Profile for the specified connection. Their conversation characteristics are defined in the single TPN profile specified in the `tpn_profile_name` field. These names are not converted from ASCII to EBCDIC.

`tpn_mask` A mask that indicates which of the specified TPNs are registered. The least significant bit (bit 0) corresponds to the first TPN (`tpn_list[0]`), and so on. The values of the bits are:

`0` TPN is not registered.

`1` TPN is registered.

If none were registered, the result is (–1) or `0x7FFFFFFF`.

attr_str Structure

This structure receives output from the GET_ATTRIBUTE request for the **snactl** and **ioctl** subroutines. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. The structure appears as follows:

```
struct attr_str
{
    long rid;
    char own_fully_qualified_lu_name[18];
    char ptner_fully_qualified_lu_name[18];
    char conversation_correlator[18];
    char modename[9];
    unsigned conn_status          :  1;
    unsigned sync_level           :  2;
    unsigned recovery_level       :  2;
    unsigned rsvd                 :  3;
    long conv_group_id
} ;
```

The **attr_str** structure parameters have the following meaning:

`rid` Specifies the variable that contains the resource ID of the GET_ATTRIBUTE conversation. This is the resource ID returned by the **snalloc** or **ioctl**(ALLOCATE) subroutine.

`own_fully_qualified_lu_name`
The variable where the request returns the fully qualified name of the LU at which the local transaction program is located.

`ptner_fully_qualified_lu_name`
The variable where the request returns the fully qualified name of the LU at which the remote transaction program is located.

`conversation_correlator`
The variable where the request returns the local program's conversation correlator, an identifier that ties the current instance of the local program to the conversation.

`modename` The variable where the request returns the mode name for the session on which the conversation is allocated.

`conn_status` A variable where the request returns a value specifying the status of the connection. Valid values are:

B'0' Connected

B'1' Stopped.

`sync_level` A variable where the request returns a value specifying the level of synchronization processing being used for the conversation. Valid values are:

B'00' No synchronization processing

B'01' Confirm synchronization processing is being used.

`recovery_level`
A variable where the request returns a value specifying the level of recovery being used for the conversation. Valid values are:

B'00' No recovery level

B'01' Reconnect.

`rsvd` This field is not used.

`conv_group_id`
A variable where the request returns the conversation group ID that identifies a specific allocated session.

confirm_str Structure

This structure receives output from the CONFIRM request for the **snactl** and **ioctl** subroutines. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. The structure appears as follows:

```
struct confirm_str
{
   long rid;
   int sense_code;
} ;
```

The **confirm_str** structure parameters have the following meanings:

`rid` Specifies the variable that contains the resource ID of the conversation to perform the confirm function. This is the resource ID returned by the **snalloc** or **ioctl**(ALLOCATE) subroutine.

`sense_code` Specifies a variable that contains the sense code returned from AIX SNA Services/6000. This parameter is used for LUs 1, 2, and 3 only.

cp_str Structure

This structure provides additional parameters that describe the cp capabilities of the adjacent node. Refer to CP–Status in the **snactl** or **ioctl** subroutine for a description of the these functions. This structure appears as follows:

```
struct cp_str
{
    long rid;
    char adj_cp_name[18];
    int conv_group_id;
    int sess_type;
    struct adj_cp_cap adj_cp_cap;
} ;
```

The **cp_str** structure parameters have the following meaning:

`rid` The variable that contains the resource ID returned from the allocate function.

`adj_cp_name` The variable where the request returns the adjacent control point (CP) name.

`conv_group_id`
The variable where the request returns the conversation group ID that identifies a specific allocated session.

`sess_type` The variable where the request returns the session type:

0 No session allocated

1 Contention Loser session

2 Contention Winner session

```
struct adj_cp_cap
    {
    unsigned  locate_gds            : 1;
    unsigned  directory_svc         : 1;
    unsigned  resource_reg          : 1;
    unsigned  char_reg              : 1;
    unsigned  topology_update       : 1;
    unsigned  cp_msu                : 1;
    unsigned  unsol_cp_msu          : 1;
    unsigned  parallel_cp           : 1;
    int flow_reduction_seq_num;
    } ;
```

The **adj_cp_cap** structure parameters have the following meaning:

`locate_gds` Indicates whether the remote cp accepts LOCATE/CDINIT requests for resources that cp controls. The value returned is either TRUE (1) or FALSE (0).

`directory_svc`
Indicates whether the remote cp forwards LOCATE/CDINIT search requests for resources to other cps. The value returned is either TRUE (1) or FALSE (0).

`resource_reg`
Indicates whether the remote cp accepts register requests. The value returned is either TRUE (1) or FALSE (0).

`char_reg` Indicates whether the remote cp accepts register requests that include resource characteristics. The value returned is either TRUE (1) or FALSE (0).

`topology_update`
Indicates whether the remote cp accepts topology database updates on the current session. The value returned is either TRUE (1) or FALSE (0).

`cp_msu` Indicates whether the remote cp accepts requests for management services data in a CP–MSU and replies to requests in a CP–MSU. The value returned is either TRUE (1) or FALSE (0).

`unsol_cp_msu`
Indicates whether the remote cp accepts unsolicited requests for management services data in a CP–MSU and replies to requests in a CP–MSU. The value returned is either TRUE (1) or FALSE (0).

`parallel_cp` Indicates whether the remote cp supports and activates parallel CP–CP sessions. The value returned is either TRUE (1) or FALSE (0).

`flow_reduction_seq_num`
: The value the remote node last used to reduce the number of database updates sent when two nodes were reconnected.

deal_str Structure

This structure provides additional parameters for the **snadeal** and **ioctl**(DEALLOCATE) subroutines. Refer to the **snadeal** or **ioctl** subroutine for a description of the these functions. This structure is as follows:

```
struct deal_str
{
    long  rid;
    unsigned  type              :  3;
    unsigned  deal_flag :  1;
    unsigned  rsvd              : 12;
    int  sense_code;
} ;
```

The **deal_str** structure parameters have the following meaning:

`rid`
: Specifies the variable that contains the resource ID of the conversation to be deallocated. This is the resource ID returned by the **snalloc** or **ioctl**(ALLOCATE) subroutine.

`type`
: Specifies the type of deallocation to be performed for this conversation. This parameter is optional. If you do not provide a value, the system uses a value of B'000'. Because this value cannot be used for LUs 1, 2, or 3, you must specify a *type* value when using those logical unit specifications. Values for type are:

  DEFAULT (B'000')
  : Use the default value specified in the sync_level parameter of the **snalloc** or **ioctl**(ALLOCATE) subroutine that established this conversation. Do not use this value for LUs 1, 2, or 3.

  DEAL_CONFIRM (B'001')
  : Perform CONFIRM logic (see the **snactl**(CONFIRM) or **ioctl**(CONFIRM) subroutine). If that request is successful, deallocate the conversation normally; otherwise, the conversation remains allocated. Do not use this value for LUs 2 or 3.

  DEAL_ABEND (B'010')
  : Deallocate the conversation abnormally. Do not use this value for LUs 1, 2, or 3.

  DEAL_FLUSH (B'011')
  : Flush the send buffer and deallocate the conversation normally.

`deal_flag`
: Specifies whether the resource ID is discarded or retained when the conversation is deallocated. This parameter is optional. If you do not provide a value, the system uses a value of B'0'. Values for deal_flag are:

  DISCARD (B'0')
  : Specifies that the resource ID be discarded. The local transaction program will not reconnect to the conversation.

RETAIN (B'1')
Specifies that the resource ID be retained. The local transaction program plans to reconnect to the conversation. Do not use this value with LUs 1, 2, and 3.

`rsvd` This field is not used.

`sense_code` Specifies a variable that contains the sense code returned from AIX SNA Services/6000. This parameter is used for LUs 1, 2, and 3 only.

erro_str Structure

This structure provides additional parameters for the SEND_ERROR request for the **snactl** and **ioctl** subroutines. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. The structure appears as follows:

```
struct erro_str
{
    long rid;
    int  sense_code;
    unsigned type  :  1;
    unsigned rsvd  : 15;
} ;
```

The `rid` parameter is the only parameter used for a mapped conversation.

`rid` Specifies the variable that contains the resource ID of the conversation to perform the send_error function. This is the resource ID returned by the **snalloc** or **ioctl**(ALLOCATE) subroutine.

The additional parameters for a basic conversation have the following meanings:

`type` Specifies the level of error being reported, as follows:

B'0' An application program produced the error being reported.

B'1' LU services Transaction Program produced the error being reported.

This parameter is optional. If not specified, SNA services provides a value of B'0'. This parameter is used for LU 6.2 Basic conversations only.

`sense_code` Specifies a variable that contains the sense code to be reported to the remote session. This parameter is used for LUs 1, 2, and 3 only.

`rsvd` This field is not used.

ext_io_str Structure

This structure provides additional input and output parameters for the **readx** and **writex** subroutines. Neither subroutine uses all parameters in the structure. Refer to the **readx** and **writex** subroutines for a description of the functions provided and the fields used.

The structure appears as follows:

```
struct ext_io_str
{
    struct input
    {
        unsigned priority              :  2;
        unsigned tpn_option            :  2;
        unsigned confirm               :  1;
        unsigned deallocate            :  1;
        unsigned deallo_type           :  3;
        unsigned deallo_flag           :  1;
        unsigned allocate              :  1;
        unsigned fill                  :  1;
        unsigned mc_gds                :  1;
        unsigned sess_type             :  1;
        unsigned flush_flag            :  2;
    } input;
    struct output
    {
        unsigned rq_to_snd_rcvd        :  1;
        unsigned what_data_rcvd        :  3;
        unsigned what_control_rcvd     :  5;
        unsigned usr_trunc             :  1;
        unsigned rsvd3                 :  6;
        unsigned gdsid                 : 16;
        int sense_code;
    } output;
    long rid;
    int usrhdr_len;
} ;
```

The **ext_io_str** structure parameters have the following meanings:

Input Parameters

These parameters are sent to SNA.

`priority` Specifies the priority option which selects a mode profile to be used to establish an appropriate session for the conversation. The priority option should be used with the **writex** subroutine only and should not be used with LU 1, 2, or 3. Priority options are:

- B'00' Use the first mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.
- B'01' Use the second mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.
- B'10' Use the third mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.

B'11' Use the fourth mode profile listed in the mode list profile for the connection. The mode list profile name is specified in the connection profile.

`tpn_option` Specifies the remote transaction program name (RTPN) option which selects a remote RTPN profile to be used to establish an appropriate session for the conversation. The following RTPN options are used by the **writex** subroutine only:

B'00' Use the first RTPN profile listed in the RTPN list profile for the connection. The RTPN list profile name is specified in the connection profile.

B'01' Use the second RTPN profile listed in the RTPN list profile for the connection. The RTPN list profile name is specified in the connection profile.

B'10' Use the third RTPN profile listed in the RTPN list profile for the connection. The RTPN list profile name is specified in the connection profile.

B'11' Use the fourth RTPN profile listed in the RTPN list profile for the connection. The RTPN list profile name is specified in the connection profile.

`confirm` This parameter is used by the **writex** subroutine only and designates whether to flush the send buffer and wait for confirmation of receipt of the data from the remote application program.

B'0' Do not issue a CONFIRM.

B'1' Issue a CONFIRM.

`deallocate` This parameter is used by both the **writex** and **readx** subroutines and designates whether to deallocate the conversation after transmitting the data associated with this subroutine:

B'0' Do not deallocate the conversation.

B'1' Deallocate the conversation. If used with an SSCP–LU application program, it could also terminate the associated LU–LU session.

`deallo_type` This parameter specifies the type of deallocation to perform when a deallocation is performed along with the subroutine:

B'000' Deallocate the conversation as specified in the sync_level parameter used in the **ioctl**(ALLOCATE) subroutine request that established this conversation. Used by the **writex** subroutine only.

B'001' Issue a CONFIRM request. If that request is successful, deallocate the conversation normally; otherwise, the conversation remains allocated. Used by the **writex** subroutine only. Do not use this value for LUs 1, 2, or 3.

B'010' Deallocate the conversation abnormally. Used by the **writex** and **readx** subroutines only.

B'011' Flush the send buffer when the conversation is in the send state and deallocate the conversation normally. Used by the **writex** subroutine only.

`deallo_flag` Specifies whether the resource ID is discarded or retained when the conversation is deallocated. Used by the **writex** and **readx** subroutines only. Values for deallo_flag are:

DISCARD (B'0')
: Specifies that the resource ID be discarded. The local transaction program will not reconnect to the conversation.

RETAIN (B'1')
: Specifies that the resource ID be retained. The local transaction program plans to reconnect to the conversation. Do not use this value with LUs 1, 2, and 3.

`allocate` This parameter specifies whether to allocate a conversation along with the subroutine. Used by the **writex** and **readx** subroutines only:

B'0' Do not allocate a conversation. The `rid` field must be supplied.

B'1' Allocate a conversation. If the `rid` parameter is 0, allocate a new conversation. If the `rid` parameter is not 0, reconnect a previous conversation identified by the value of rid.

The `allocate` parameter can be used with the `deallocate` parameter (but not for LUs 1, 2, or 3). If deallocate is also set, the **readx** and **writex** subroutines perform the following actions:

1. Allocates a conversation as described above.
2. Transfers the data associated with the subroutine.
3. Deallocates the conversation.

`fill` Specifies whether the program receives data without regard to the logical record format of the data. This parameter is optional and is used by the **readx** subroutine only. If you do not specify one of the two following values, the program uses a value of B'0'. Always use a value of B'0' for mapped conversations.

BUFFER (B'0')
: Specifies that the program receives data without regard to the logical record format of the data.

LL (B'1')
: Specifies that the program receives one complete logical record, or a logical record that has been truncated to the length specified in the *length* parameter of this subroutine. Do not use with LU 1, 2, or 3.

`mc_gds` Reserved for use by mapped conversation RTS.

`sess_type` Specifies the type of session to be allocated if the allocate field indicates that a session should be allocated. Used for LU 1, 2, and 3. Valid values are:

B'1' SSCP–LU session

B'0' LU–LU session.

`flush_flag` This parameter indicates whether to perform a FLUSH (of the LUs send buffers) request in addition to the requested **writex** operation.

I_O_NO_FLUSH (B'00')
: Do not perform the **flush**.

I_O_FLUSH_NOT_EC (B'01')
: Perform the **flush**, but do not indicate end of chain.

I_O_FLUSH_EC (B'10')
: Perform the **flush**, indicating end of chain. This function is used by LUs 1, 2, and 3 only.

Output Parameters

These parameters are set by SNA.

`rq_to_snd_rcvd`
: Indicates whether a request to send has been received. The `rq_to_snd_rcvd` parameter specifies the variable used by the **writex** and **readx** subroutines only:

B'0'
: A request to send has not been received from the remote transaction program.

B'1'
: A request to send has been received from the remote transaction program.

`what_data_rcvd`
: Specifies the variable that gets set to indicate what type of data the program received. Used by the **readx** subroutine only:

DATA (B'000')
: Indicates that data has been received by the program. Occurs only when the *fill* parameter for this call is **buffer**. Not used for LUs 1, 2, or 3.

DATA_COMP (B'001')

LU 6.2: Indicates that a complete logical record, or the last remaining portion of a logical record, has been received by the program. Occurs only when the *fill* parameter for this call is **ll**.

LUs 1, 2, 3: Indicates that a complete chain element was received.

DATA_INCOMP (B'010')

LU 6.2: Indicates that less than a complete logical record has been received by the program. Occurs only when the *fill* parameter for this call is **ll**.

LUs 1, 2, 3: Indicates that a complete chain element was not received.

LL_TRUNCATED (B'011')
: Indicates that the 2–byte ll field of a logical record was truncated after the first byte and that the LU has discarded the ll field. The program does not receive the ll field. Not used for LUs 1, 2, or 3.

FMH_COMPLETE (B'100')
: Indicates that the data received was FM header data for an LU 1 session and that the complete header has been received.

FMH_INCOMPLETE (B'101')
Indicates that the data received was FM header data for an LU 1 session and that the complete header has not been received.

B'110' Not used.

B'111' Not used.

`what_control_rcvd`
Specifies the variable that is set to indicate the type of control that the program received. Used by the **readx** subroutine only:

B'X0000'
No control information was received.

SEND (B'X0001')
Indicates that the remote program has entered the receive state, placing the local program in the send state.

CONFIRM (B'X0010')
Indicates that the remote program issued a CONFIRM request. The local program must respond with an **ioctl** subroutine, using either a CONFIRMED request or a SEND_ERROR request.

CONFIRM_SEND (B'X0011')
Indicates that the remote program used an **ioctl** or **snactl** subroutine to issue a PREPARE_TO_RECEIVE request, and that the type parameter was set to B'10' (confirm).

CONFIRM_DEALLOCATE (B'X0100')
Indicates that the remote program used an **ioctl** or **snactl** subroutine to issue a DEALLOCATE request with the type parameter set to B'001' (confirm). Not used for SSCP–LU sessions for LUs 1, 2, and 3.

NORMAL_DEALLOCATE (B'X0101')
Indicates that the remote program issued a DEALLOCATE request with the type parameter set to B'011' (flush). Not used for SSCP–LU sessions for LUs 1, 2, and 3.

CONFIRM_DEALLOCATE_RETAIN (B'X0110')
Indicates that the remote program issued a DEALLOCATE request with the type parameter set to B'001' (confirm) and the deal_flag parameter set to retain. Not used for LUs 1, 2, and 3.

NORMAL_DEALLOCATE_RETAIN (B'X0111')
Indicates that the remote program issued a DEALLOCATE request with the type parameter set to B'011' (flush) and the deal_flag parameter set to retain. Not used for LUs 1, 2, and 3.

Note: The following parameters are used by LUs 1, 2, or 3 only:

FLUSH RECEIVED (B'X1000')
Specifies end chain, RQE, not change direction.

NOT END OF DATA (B'X1001')
Specifies end chain was not received from the host.

BEGIN CHAIN (B'1XXXX')
Specifies begin chain indicator was received from the host. This may be set in addition to other returned flags.

NOT BEGIN CHAIN (B'0XXXX')
Specifies begin chain was not received in this data buffer.

Note: 'X' means that the X bit positions are meaningless for this function.

`usr_trunc` Indicates that the length of the user header field was not large enough for the received header data. You can get no information from the user header field when the user header has been truncated.

`rsvd3` This field is not used.

`gdsid` This field is used for mapped conversation runtime service (RTS) routines only.

`sense_code` Specifies the variable that is set to the value of the sense code for negative responses received. Used for LUs 1, 2, and 3 only.

`rid` This parameter specifies the resource ID returned by the **snalloc** or **ioctl**(ALLOCATE) subroutine for this connection. This parameter has the following effects:

- For the **writex** subroutine with the allocate bit on and rid equal to 0, the system allocates a new conversation and returns the resource ID in rid.
- For the **writex** subroutine with the allocate bit on and rid not equal to 0, the system reconnects an old conversation identified by the value in rid.
- For the **readx** subroutine with the allocate bit on and rid not equal to 0, it indicates a remotely attached allocation.

`usrhdr_len` This parameter specifies the number of bytes of header data to be sent or that was received with the data. The header data is provided in the usrhdr field (see "User Header Field" for the **writex** subroutine or "User Header Field" for the **readx** subroutine.).

flush_str Structure

This structure provides additional parameters for the FLUSH request for the **snactl** and **ioctl** subroutine. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. The structure appears as follows:

```
struct flush_str
{
      long rid;
      unsigned end_chain      :  1;
      unsigned rsvd           : 15;
      int sense_code;
} ;
```

The **flush_str** structure parameters have the following meanings:

`rid` This parameter specifies the resource ID returned by the **ioctl**(ALLOCATE) or the **snalloc** subroutine for this connection.

`end_chain` Specifies whether or not to send the buffer with the *end chain* indication. The program specifies this parameter as a 1 to complete a chain. To flush

the send buffer without completing the chain, the program specifies this parameter as a 0. Valid values are:

0 Send buffer without *end chain.*

1 Send buffer with *end chain.*

`rsvd` This field is not used.

`sense_code` Specifies a field that receives indications of errors that occurred on previously sent data. Used for LU 1, 2, and 3 only.

fmh_str Structure

This structure used for LU 1, 2, and 3 provides additional parameters for the SEND_FMH request for the **snactl** and **ioctl** subroutines. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. The structure appears as follows:

```
struct fmh_str
{
      long rid;
      short fmh_len;
      unsigned type           :  2;
      unsigned rsvd           : 14;
      char *fmh_addr;
      int sense_code;
} ;
```

The **fmh_str** structure parameters have the following meanings:

`rid` Specifies the variable that contains the resource ID of the conversation to perform the SEND_FMH request. This is the resource ID returned by the **snalloc** or **ioctl**(ALLOCATE) subroutine.

`fmh_len` Specifies the length (in bytes) of the FM header to be sent.

`type` Specifies the type of request to be performed for this conversation. Values for type are:

B'00' Flush without end chain

B'01' Flush the FMH with end chain.

B'10' Execute a CONFIRM function (see the **snactl**(CONFIRM) subroutine).

B'11' Do not flush the FMH.

`rsvd` This field is not used.

`fmh_addr` Specifies a pointer to the address of the FM header to be sent.

`sense_code` Specifies a variable that contains the sense code returned from the AIX SNA Services/6000.

get_parms Structure

Returns data associated with the **get_parameters** structure of the **ioctl** subroutine.

```
struct get_parms
{
      int gparms_size;
      char gparms_data [MAX_GETPARMS_DATA];
} ;
```

The **get_parms** structure parameters have the following meanings:

`gparms_size` Specifies the length (in bytes) of the parameter data returned in the `gparms_data` field.

`gparms_data` Contains the following data:

TPN Name
: Specifies the transaction program name, which has a maximum length of 64 bytes plus a terminating blank.

Connection Name
: Specifies the connection name, which has a maximum length of 15 bytes plus a terminating blank.

RID
: Specifies the resource ID, which has a maximum length of 6 bytes plus a terminating blank.

PIP
: Specifies the program initialization parameters, which have a maximum of sixteen fields, each having a maximum of 64 bytes plus a terminating blank.

gstat_str Structure

This structure provides current link and session information in response to a GET_STATUS request with either an **ioctl** or **snactl** subroutine. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. This structure is defined for use by LUs 1, 2, and 3 only. The structure appears as follows:

```
struct gstat_str
{
      int sscp_sense_code;
      int status;
      unsigned  rtn_image          :  1;
      unsigned  rsrv               : 15;
      short  image_len;
      char  *image;
      int lu_sense_code;
} ;
```

The **gstat_str** structure parameters have the following meaning:

`status` Specifies the current status of the physical and logical link and the SSCP-LU and LU-LU sessions. Several values may be set at once (bit settings). Valid values ( ) for this parameter are expressed in hexadecimal notation below:

LINK_ACTIVE (0X1)
: The link is active.

LINK_INACTIVE (0X2)
The link is not active.

LINK_TIMEOUT (0X4)
A link timeout has occurred.

SSCP_LU_ACTIVE (0X8)
An SSCP–LU session is active on the link.

SSCP_LU_INACTIVE (0X10)
An SSCP–LU session is allocated to the link, but is not active.

LU_LU_ACTIVE (0X20)
An LU–LU session is allocated to the link and is currently active.

LU_LU_INACTIVE (0X40)
An LU–LU session is allocated to the link, but is not active.

LU_LU_RESET (0X80)
The LU–LU session allocated to the link has been reset.

HOST_BID (0X100)
A BID, request to begin a Bracket, was received from the host.

LU_LU_SHUTDOWN (0X200)
The LU–LU session has been shut down.

LU_LU_CONFIRM_RCVD (0X400)
A CONFIRM request is received from the host on the LU–LU session.

SSCP_LU_CONFIRM_RCVD (0X800)
A CONFIRM request is received from the host on the SSCP–LU session.

RSP_TO_RTS_RCVD (0X1000)
A response is received from the host for a request to send, sent on the LU–LU session.

RQTS_RCVD (0X2000)
A request to send is received from the host.

RSP_TO_CANCEL_RCVD (0X4000)
A response is received from the host for a CANCEL request previously sent on the LU–LU session.

`rtn_image` When set, this field indicates that the BIND image associated with the session should be returned in the buffer pointed to by image_ptr.

`image_len` This field contains one of the following values if `rtn_image` is set:

- When the GET_STATUS is issued, this field contains the maximum amount of BIND image data (in bytes) that can be returned by the request.
- When the request is complete, this field contains the actual amount of BIND image data (in bytes) that was returned.

`image` Specifies a pointer (up to 26 bytes or the image length) to the buffer area in which the BIND image data is to be stored.

`sscp_sense_code`
This parameter is set to 0 if the response is positive and to the sense code received if negative.

`lu_sense_code`
This parameter is set to 0 if the response is positive and to the sense code received if negative.

pip_str Structure

This structure provides program initialization parameters (PIP) to be sent to a remote program by the **snalloc** or **ioctl**(ALLOCATE) subroutine. Refer to the **snalloc** or **ioctl** subroutine for a description of the these functions. This structure is defined for use by LU 6.2 only. The structure appears as follows:

```
struct pip_str
{
      int          sub_num;
      char         sub_data [16][65].;
} ;
```

The **pip_str** structure parameters have the following meaning:

`sub_num` A variable that specifies the number of PIP subfields in `sub_data`.

`sub_data` The array of program initialization data for the remote program. There may be up to 16 entries, each containing up to 64 bytes of initialization data.

prep_str Structure

This structure provides additional parameters for the PREPARE_TO_RECEIVE request for the **snactl** and **ioctl** subroutines. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. The structure appears as follows:

```
struct prep_str
{
      long rid;
      unsigned type      :  2;
      unsigned rsvd      : 14;
      int sense_code;
} ;
```

The **prep_str** structure parameters have the following meanings:

`rid` This parameter specifies the resource ID returned by the **ioctl**(ALLOCATE) or **snactl**(ALLOCATE) subroutine for this connection.

`type` Specifies the type of request to be performed for this conversation. Values for type are:

SYNCL_DEF (B'00')
Use the default value specified in the sync_level parameter of the **snalloc** subroutine that established this conversation.

SYNCL_NONE (B'01')
Flush the send buffer and enter the receive state.

SYNCL_CONFIRM (B'10')
: Execute a confirm function (see the **snactl**(CONFIRM) subroutine). If that request is successful, enter the receive state.

SYNCL_FLUSH (B'11')
: Flush the send buffer and enter the receive state.

`rsvd` This field is not used.

`sense_code` Specifies a field that receives indications of errors that occurred on previously sent data. This parameter is used by LUs 1, 2, and 3 only.

read_out Structure

This structure receives output from the **snaread** subroutine. Refer to the **snaread** subroutine for a description of that function. The structure appears as follows:

```
struct read_out
{
      long        rid;
      int         request_to_send_received;
      int         what_data_rcvd;
      int         what_control_rcvd;
      int         sense_code;
} ;
```

The **read_out** structure parameters have the following meanings:

`rid` Specifies the variable that contains the resource ID returned by the **snalloc** subroutine that allocated the resource to be read.

`request_to_send_received`
Specifies the variable that gets set to indicate whether a request to send has been received:

TRUE(1)
: A request to send has been received from the remote transaction program.

FALSE (0)
: A request to send has not been received from the remote transaction program.

`what_data_rcvd`
Specifies the variable that gets set to indicate what type of data the program received:

0 Indicates that data has been received by the program. Occurs only when the *fill* parameter for this subroutine is buffer.

DATA_COMPLETE (1)
: Occurs only when the *fill* parameter for this subroutine is ll. It indicates that a complete logical record, or the last remaining portion of a logical record, has been received by the program.

DATA_INCOMPLETE (2)
: Occurs only when the *fill* parameter for this subroutine is ll. It indicates that less than a complete logical record has been received by the program.

3 Indicates that the 2–byte ll field of a logical record was truncated after the first byte and that the LU has discarded the ll field. The program does not receive the ll field. Not used for LUs 1, 2, or 3.

4 Indicates that the data received was FM header data for an LU 1 session and that the complete FM header was received.

5 Indicates that the data received was FM header data for an LU 1 session, but that the complete FM header was not received.

`what_control_rcvd`
Specifies the variable that is set to indicate the type of control that the program received:

0 No control information received.

1 Indicates that the remote program has entered the receive state, placing the local program in the send state.

2 Indicates that the remote program used a snactl subroutine to issue a CONFIRM request. The local program must respond with a **snactl** subroutine using either a CONFIRMED request or a SEND_ERROR request.

3 Indicates that the remote program used a **snactl** subroutine to issue a PREPARE_TO_RECEIVE request, and that the type parameter was set to B'10' (confirm).

4 Indicates that the remote program used a **snadeal** subroutine with the type parameter set to B'001' (confirm). Not used for SSCP–LU sessions for LUs 1, 2, and 3.

5 Indicates that the remote program used a **snadeal** subroutine with the type parameter set to B'011' (flush). Not used for SSCP–LU sessions for LUs 1, 2, and 3.

6 Indicates that the remote program used a **snadeal** subroutine with the type parameter set to B'001' (confirm) and the deal_flag parameter set to retain. Not used for LUs 1, 2, and 3.

7 Indicates that the remote program used a **snadeal** subroutine with the type parameter set to B'011' (flush) and the deal_flag parameter set to retain. Not used for LUs 1, 2, and 3.

`sense_code` Specifies the variable that is set to the value of the sense code for negative responses. Used for LUs 1, 2, and 3 only.

stat_str Structure

This structure provides additional parameters for the SEND_STATUS request for the **snactl** and **ioctl** subroutines. Refer to the **snactl** or **ioctl** subroutine for a description of the these functions. The structure appears as follows:

```
struct stat_str
{
      long rid;
      unsigned type           : 4;
      unsigned id       : 8;
      int  sense_code;
} ;
```

The **stat_str** structure parameters have the following meanings:

`rid` Specifies the variable that contains the resource ID returned by the **snalloc** or **ioctl**(ALLOCATE) subroutine that performs the SEND_STATUS function.

`type` Specifies the status condition to be reported. Use one of the following values ( ) as defined in the **luxsna.h** header file:

POWER_ON (0)
: The device is on.

POWER_OFF (1)
: The device is off.

UNAVAILABLE (2)
: The device is not configured.

PERMANENT_ERROR (3)
: The device has an error which cannot be corrected.

PS_ALTERED (4)
: Presentation space altered.

UNBIND_REQUESTED (5)
: A request shutdown (RSHUTD) command has been sent to the partner LU requesting an unbind (UNBIND) command to be sent to this secondary LU. This *type* does not use the *id* field. This *type* does not cause the received data to be rejected. The application program should continue to read the data until it receives SNA_NSES (session not active).

ATTENDED (6)
: The device is attended by an operator (LU 1 only).

UNATTENDED (7)
: The device is no longer attended by an operator (LU 1 only).

`ID` Specifies the ID of the device for which status is being reported.

`sense_code` Specifies a variable that contains the sense code returned from AIX SNA Services/6000.

write_out Structure

This structure provides additional parameters for the **snawrit** subroutine. Refer to the **snawrit** subroutine for a description of these functions. The structure appears as follows:

```
struct write_out
{
      int request_to_send_received;
      int sense_code;
} ;
```

The **write_out** structure parameters have the following meanings:

`request_to_send_received`
: Specifies the variable that gets set to indicate whether a request to send has been received:

TRUE(1)
A request to send has been received from the remote transaction program.

FALSE (0)
A request to send has not been received from the remote transaction program.

`sense_code` Specifies the variable that is set to the value of the sense code for negative responses. Used for LUs 1, 2, and 3 only.

Constant Definitions

The **luxsna.h** file contains constant definitions that are used in the following areas of AIX SNA Services/6000:

- Status codes (see the **gstat_str** structure)
- Error codes
- Request codes for the **snactl** and **ioctl** subroutines.

Error Code Constants

This file defines the error return values that are exclusive to AIX SNA Services/6000. The AIX SNA Services/6000 subroutines set the **errno** global variable to one of the following values when an error occurs to indicate the nature or cause of the error.

Error Code Constants		Page 1 of 3
Name	**Code**	**Definition**
SNA_CTYPE	101	The specified conversation type does not match the indicated conversation.
SNA_NREC	103	Reconnect is not supported.
SNA_NSYC	104	Sync level is not supported.
SNA_ALFN	105	An allocation failure occurred. Do not try the operation again.
SNA_ALFR	106	An allocation failure occurred. Try the operation again.
SNA_LUNREC	107	Reconnect is not supported by the LU.
SNA_LUNSYC	108	Sync level is not supported by the LU.
SNA_RID	109	The resource ID was invalid.
SNA_STATE	110	The network management request was issued while the Program was not in an allowed state.
SNA_RFR	111	A resource failure occurred. Try the operation again.
SNA_RFN	112	A resource failure occurred. Do not try again.
SNA_PROTOCOL	113	An SNA protocol violation occurred.
SNA_NPIP	114	Remote program initialization parameter (PIP) data is not supported.
SNA_PNSYC	115	Sync level is not supported by the program.
SNA_PNREC	116	Reconnection is not supported by the program.
SNA_NRREC	117	Could not reconnect to the transaction program. Do not try again.

Error Code Constants		Page 2 of 3
Name	**Code**	**Definition**
SNA_PPURG	118	Program error purging.
SNA_PNTR	119	A program error occurred since no truncation is allowed.
SNA_PTR	120	Program error truncate.
SNA_PGMDEAL	121	A deallocation occurred due to the abnormal ending of the remote program.
SNA_BOUNDARY	122	The function was not requested on a logical record boundary.
SNA_NOMODE	123	Invalid mode name specified.
SNA_RREC	124	Cannot connect to the transaction program. Try the operation again.
SNA_NOCONN	125	The SNA connection has been stopped.
SNA_NRESTART	126	A recovery_level value of restart is not valid for this subroutine.
SNA_NOTPN	127	The specified transaction program name is not valid.
SNA_NRMDEAL	129	A normal deallocation terminated the conversation.
SNA_SVCDEAL	130	A deallocation occurred due to an abnormal ending of an system service (systems logic error).
SNA_TIMDEAL	131	Deallocate abend due to excessive time having elapsed.
SNA_WRGPIP	132	The remote program initialization data (PIP) specified was not correct.
SNA_INVACC	133	Access security information invalid.
SNA_SPURG	134	A SVC error occurred; purging.
SNA_SNTR	135	A service transaction program error occurred; no truncate.
SNA_STR	136	A service transaction program error occurred; truncate.
SNA_NDELAY	137	Delay allocation not supported.
SNA_SVCTYPE	138	An unsupported type was specified.
SNA_NFMH	139	The FM Header data gds variable is not supported by mapped conversation.
SNA_NMAPPING	140	The MAP name is not supported by mapped conversation.
SNA_MAP_NOT_FOUND	141	Map name not found.
SNA_MAPEXEC	142	Map execution failure.
SNA_GDSID	143	Invalid GDS identifier in data.
SNA_SHUT	144	A shutdown request was received.
SNA_NSES	145	The session is not established (LUs 1, 2, 3, and 6.2) or not active (LUs 1, 2, and 3).

Error Code Constants		Page 3 of 3
Name	**Code**	**Definition**
SNA_PARMS	146	Input parameters not valid.
SNA_NTPN	147	Transaction Program can not be started, no retry.
SNA_NTPR	148	Transaction Program can not be started due to lack of resources, retry.
SNA_RCANC	149	Received cancel for LU 1, 2, or 3.
SNA_SENSE	150	Sense code available for LU 1, 2, or 3 (exception request or negative response received).
SNA_NOTCP	151	This is not a CP connection.
SNA_FAIL	160	There was an SNA system failure.
SNA_NSACT	161	No session can be started as the session limit is set to 0.
SNA_NSLMT	162	No session can be activated as the number of sessions of the requested type has been exceeded.
SNA_INOP	164	Link INOP received.
SNA_HIER_RESET	165	Hierarchical reset received.
SNA_NO_LU	166	No LUs registered for generic SNA.
SNA_INUSE	170	The session between the system services control point and the physical unit is being used by another application.
SNA_NOTAVAIL	171	The requested session between the system services control point and the physical unit was not available.
SNA_UNDEF_SVR	172	The application server is not defined.
SNA_INVALID	173	The ID specified for the session between the system services control point and the physical unit (SSCP_ID) is not valid.
SNA_LENGTH	174	The length specified for the NMVT data is not valid.
SNA_ERP	175	The physical unit is not active. An error recovery procedure (ERP) instructing you to activate the physical unit (ACTPU) was received.
SNA_INACT	176	The session between the system services control point and the physical unit is inactive.

Request Code Constants

This file defines the following constants and their codes for use in the *request* parameter of the **ioctl** or **snactl** subroutine.

Request Code Constants		
Name	**Code**	**Definition**
ALLOCATE	1	Allocates a conversation. Used by the **ioctl** subroutine only.
DEALLOCATE	2	Deallocates a conversation. Used by the **ioctl** subroutine only.
CONFIRM	3	Sends a request for confirmation of transmission to the remote transaction program.
CONFIRMED	4	Positive response to a CONFIRM request.
FLUSH	5	Transmits everything in the send buffer to the remote transaction program. Used for LU 1, 2, or 3 only.
PREPARE_TO_RECEIVE	6	Changes the conversation direction to allow the local transaction program to receive.
HIER_RESET_RSP	6	Hierarchical reset response for Generic SNA Applications.
REQUEST_TO_SEND	7	Request to change the conversation direction to allow the local transaction program to send.
INOP_RSP	7	Link inoperative response for Generic SNA Applications.
SEND_FMH	8	Sends the FM header to the remote LU. Used by the **snactl** subroutine for LU 1 on a basic conversation only. Used for LU 1, 2, or 3 only.
SEND_ERROR	9	Negative response to a CONFIRM request or incorrect data received.
GET_ATTRIBUTE	10	Gets information about the specified LU 6.2 conversation.
SEND_STATUS	11	Sends status information about the devices on the local session (LUs 1, 2, and 3, only) to the host program.
GET_STATUS	12	Gets information about the current link and session on a basic conversation only. Used for LUs 1, 2, or 3 only.
CP_STATUS	13	Requests the control point name, the session type, and the control point capabilities of the remote node.
ALLOCATE_LISTEN	14	Registers a list of transaction program names (TPNs) for which an application wishes to accept allocate requests. Used for LU 6.2 only.
GET_PARAMETERS	15	Retrieves the data associated with the receipt of an allocate request for a registered TPN on a particular connection. The GET_PARAMETERS argument is used in conjunction with the ALLOCATE_LISTEN argument. Used for LU 6.2 only.

File

/usr/include/luxsna.h The path to the **luxsna.h** header file.

Sockets in.h File

Purpose

Defines Internet constants and structures.

Description

The **in.h** file contains Internet system constants and socket structures required for inclusion in programs using socket subroutines. The full pathname for the **in.h** file is **/usr/include/netinet/in.h**. The **usr/include/netinet/in.h** file contains data definitions for socket types, address families, and options.

Protocols

IPPROTO_IP	A dummy for IP.
IPPROTO_ICMP	A control message protocol.
IPPROTO_GGP	A gateway^2 (deprecated)
IPPROTO_TCP	TCP.
IPPROTO_EGP	The exterior gateway protocol.
IPPROTO_PUP	PUP.
IPPROTO_UDP	The user datagram protocol.
IPPROTO_IDP	XNS IDP.
IPPROTO_RAW	The raw IP packet.
IPPROTO_MAX	

Link Numbers

IMPLINK_IP

IMPLINK_LOWEXPER

IMPLINK_HIGHEXPER

Internet Address

```
struct in_addr{
u_long s_addr;
};
```

Bits in Internet Address Integers

IN_CLASSA(i)	(((long) (i) & 0x80000000) == 0)
IN_CLASSA_NET	0xff000000
IN_CLASSA_NSHIFT	24
IN_CLASSA_HOST	0x00ffffff
IN_CLASSA_MAX	128
IN_CLASSB(i)	(((long) (i) & 0xc0000000) == 0x80000000)
IN_CLASSB_NET	0xffff0000
IN_CLASSB_NSHIFT	16
IN_CLASSB_HOST	0x0000ffff

IN_CLASSB_MAX	65536
IN_CLASSC (i)	(((long) (i) & 0xe0000000) == 0xc0000000)
IN_CLASSC_NET	0xffffff00
IN_CLASSC_NSHIFT	8
IN_CLASSC_HOST	0x000000ff
IN_CLASSD(i)	(((long) (i) & 0xf0000000) == 0xe0000000)
IN_MULTICAST(i)	IN_CLASSD(i)
IN_EXPERIMENTAL(i)	(((long) (i) & 0xe0000000) == 0xe0000000)
IN_BADCLASS(i)	(((long) (i) & 0xf0000000) == 0xf0000000)
INADDR_ANY	(u_long) 0x00000000)
INADDR_BROADCAST	(u_long) 0xffffffff
KERNAL	
INADDR_NONE	0xffffffff
INLOOPBACKNET	127

Socket Address

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
struct in_addr sin_addr;
         char     sin_zero[8};
};
```

Implementation Specifics

The **/usr/include/netinet/in.h** file is part of AIX Base Operating System (BOS) Runtime.

All applications including the **/usr/include/netinet/in.h** file must be compiled with **_BSD** defined. In addition, when applicable, all socket applications must include the BSD library **libbsd**.

Related Information

Additional header files that contain a **sockaddr** structure are the **/usr/include/sys/socket.h** file and the **/usr/include/sys/un.h** file.

Sockets nameser.h File

Purpose

The **/usr/include/arpa/nameser.h** file contains Internet name server information.

Description

The **usr/include/arpa/nameser.h** file contains Internet constants, data definitions, and socket structures required for inclusion in programs using socket subroutines.

Internet Constants

PACKETETSZ The maximum packet size.

MAXDNAME The maximum domain name.

MAXCDNAME The maximum compressed domain name.

MAXLABEL The maximum length of domain label.

QFIXEDSZ Number of bytes of fixed size data in query structure.

RRFIXEDSZ Number of bytes of fixed size data in resource record.

Internet Nameserver Port Number

NAMESERVER_PORT 53

Currently Defined Opcodes

QUERY The standard query.

IQUERY The inverse query.

STATUS The nameserver status query.

xxx Non standard.

UPDATEA Add resource record.

UPDATED Delete a specific resource record.

UPDATEDA Delete all named resource records.

UPDATEM Modify a specific resource record.

UPDATEMA Modify all named resource records.

ZONEINIT Initial zone transfer.

ZONEREF Incremental zone refresh.

Currently Defined Response Codes

NOERROR No error.

FORMERR Format error.

SERVFAIL Server failure.

NXDOMAIN Non existent domain.

NOTIMP Not implemented.

REFUSED Query refused.

NOCHANGE Update failed to change db.

Structure

```
typedef struct { u_short id; /* query identification number */ #ifd
efined (sun) || defined (sel)    || defined (pyr) || defined (is68k)
 \ ||defined (tahoe) || defined (aiws) || defined
   (BIT-ZERO-ON-LEFT)

 /* bit zero on left:  Gould and similar architectures   */
                      /* fields in third byte   */
   u_char  qr:1;      /* response flag        */
   u_char  opcode:4;  /* purpose of message   */
   u_char  aa:1;      /* authoritive answer   */
   u_char  tc:1;    /*  truncated message    */
   u_char  rd:1;      /* recursion desired fields in fourth byte  */
   u_char  ra:1;      /* recursion available   */
   u_char  pr:1;      /* primary server required (non standard) */
   u_char  unused:2;  /* unused bits   */
   u_char  rcode:4;   /* response code   */

   #else
   #if defined (vax) || defined(ns32000) || defined
  (BIT_ZERO_ON_RIGHT)
   /* bit zero on right:  VAX */


  /* fields in third byte   */
   u_char  rd:1;      /* recursion desired   */
   u_char  tc:1;      /* truncated message   */
   u_char  aa:1;      /* authoritive answer   */
   u_char  opcode:4;  /* purpose of message   */
   u_char  qr:1;      /* response flag   */

  /* fields in fourth byte   */
   u_char  rcode:4;   /* response code   */
   u_char  unused:2;   /*unused bits   */
   u_char  pr:1;      /* primary server required (non standard) */
   u_char  ra:1;      /* recursion available   */

  #else
   /* You must determine what the correct bit order is for your
      compiler  */
   UNDEFINED_BIT_ORDER;
   #ENDIF
   #ENDIF

  /* remaining bytes   */
   u_short  qdcount;   /* number of question entries   */
   u_short  ancount;   /* number of answer entries   */
   u_short  nscount;   /* number of authority entries   */
   u_short  arcount;   /* number of resource entries   */
   } HEADER;
```

Compressed Domain Names

Definitions for handling compressed domain names

INDIR_MASK 0xc0

rrec Structure

The **rrec** structure is used for passing resource records around.

struct rrec {

```
short  r_zone     zone number.
short  r_class    class number.
short  r_type     Type number.
u_long r_ttl      Time to live.
int    r_size     Size of data area.
char   r_data     Pointer to data.
};
extern  u_short  _getshort ( );
extern  u_long   _getlong ( );
```

Implementation Specifics

The **/usr/include/arpa/nameser.h** file is part of AIX Base Operating System (BOS) Runtime.

All applications including the **/usr/include/arpa/nameser.h** file must be compiled with **_BSD** defined. In addition, when applicable, all socket applications must include the BSD library **libbsd**.

Related Information

Domain name access subroutines using the **nameser.h** file are the **res_mkquery** subroutine, **res_init** subroutine and **res_send** subroutine.

Domain name translation subroutines using the **nameser.h** file are the **dn_comp** subroutine, **dn_expand** subroutine, **dn_find** subroutine, and **dn_skipname** subroutine.

Byte stream and byte boundary retrieval subroutines using the **nameser.h** file are the **_getlong** subroutine, **_getshort** subroutine, **putlong** subroutine, and **putshort** subroutine.

Sockets Overview and Understanding Header Files in *Communications Programming Concepts*.

Sockets netdb.h File

Purpose

Defines the structures returned by the network data base library.

Description

The **/usr/include/netdb.h** header file contains structures that are used globally by socket library subroutines. The **/usr/include/netdb.h** file contains the following structures:

- **hostent** structure
- **netent** structure
- **protoent** structure
- **servent** structure.

hostent Structure

The **hostent** structure is defined in the **/usr/include/netdb.h** header file and 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 0 (zero), of pointers to the network addresses for the host. Host addresses are returned in network byte order.

h_addr The first address in the **h_addr_list** member, provided for backward compatibility

netent Structure

The **netent** structure is defined in the **netdb.h** header file and contains the following members:

```
char    *n_name;        /* official name of net */
char    **n_aliases;    /* network alias list   */
int     n_addrtype;     /* net number type      */
long    n_net;          /* net number           */
```

The members of the structure are defined as follows:

n_name Official name of the network.

n_aliases An array, terminated with a 0 (zero), of alternate names for the network.

n_addrtype The type of network number being returned. **AF_INET** is the only valid value for this field.

n_net The network number. Network numbers are returned in machine–byte order.

protoent Structure

The **protoent** structure is defined in the **/usr/include/netdb.h** header file and 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 as follows:

p_name Official name of the protocol

p_aliases An array, terminated by the number 0, of alternate names for the protocol

p_proto The protocol number.

servent Structure

The **servent** (service entry) structure is defined in the **/usr/include/netdb.h** header file and 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 as follows:

s_name Official name of the service.

s_aliases An array of alternate names for the service. Terminate the array with a value of 0.

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.

Implementation Specifics

The **/usr/include/netdb.h** file is part of AIX Base Operating System (BOS) Runtime.

All applications including the **/usr/include/netdb.h** file must be compiled with **_BSD** defined. In addition, when applicable, all socket applications must include the BSD library **libbsd**.

Related Information

Host information retrieval routines using the **/usr/include/netdb.h** file are the **gethostent** subroutine, **endhostent** subroutine, **gethostbyname** subroutine, **gethostbyaddr** subroutine, and **sethostent** subroutine.

Network information retrieval subroutines using the **/usr/include/netdb.h** file are the **endnetent** subroutine, **getnetbyname** subroutine, **getnetbyaddr** subroutine, **getnetent** subroutine, and **setnetent** subroutine.

Service information retrieval subroutines using th **/usr/include/netdb.h** file are the **endservent** subroutine, **getservbyname** subroutine, **getservent** subroutine, **getservbyport** subroutine and **setservent** subroutine.

Protocol information retrieval subroutines using the **/usr/include/netdb.h** file are the **endprotoent** subroutine, **getprotobynumber** subroutine, **getprotobyname** subroutine, **getprotoent** subroutine, and **setprotoent** subroutine.

Sockets resolv.h File

Purpose

Contains global definitions and variables used by resolver subroutines.

Description

The **/usr/include/resolv.h** file contains the **_res** data structure and defines the options to be used in the **_res** option field.

_res Data Structure

The **_res** data structure contains the following members:

retrans Specifies retransmission time interval.

retry Specifies the number of times to retransmit.

options Specify the resolver options that can be logically ORed.

nscount Specifies the number of name servers.

sockaddr_in nsaddr_list [MAXNS]
Specifies the address of a name server.

nsaddr_list [0] Provided for backward compatibility.

id Specifies current packet id.

defdname [MAXDNAME]
Specifies the default domain.

dnsrch [MAXDNSRCH+1]
Specifies the components of the domain to search.

Resolver Options

The **/usr/include/resolv.h** file contains a data structure termed **_res** that is used by the resolver subroutines. The **_res** structure contains an options field that is constructed by logically ORing the following values:

RES_INIT Indicates whether the initial name server and default domain name have been initialized (that is, whether **res_init** has been called).

RES_DEBUG Prints debugging messages.

RES_USEVC Uses TCP connections for queries instead of UDP.

RES_STAYOPEN
Used with **RES_USEVC**, keeps the TCP connection open between queries. While UDP is the mode normally used, TCP mode and this **option** are useful for programs that regularly perform many queries.

RES_RECURSE
Sets the recursion desired bit for queries. This is the default.

Note: The **res_send** subroutine does not perform iterative queries and expects the name server to handle recursion.

RES_DEFNAMES
Appends the default domain name to single label queries. This is the default.

QUERY	Standard query.
IQUERY	Inverse query.
CQUERYM	Completion query (multiple).
CQUERYU	Completion query (unique).

Files

/etc/resolv.conf	Defines DOMAIN nameserver information for local resolver routines.

Implementation Specifics

The **/usr/include/resolv.h** file is part of AIX Base Operating System (BOS) Runtime.

All applications including the **/usr/include/resolv.h** file must be compiled with **_BSD** defined. In addition, when applicable, all socket applications must include the BSD library **libbsd**.

Related Information

Resolver subroutines are the **res_mkquery** subroutine, **res_init** subroutine, **res_send** subroutine, **dn_comp** subroutine, **dn_expand** subroutine, **dn_find** subroutine, **dn_skipname** subroutine, **_getlong** subroutine, **_getshort** subroutine, **putlong** subroutine, and **putshort** subroutine.

Sockets socket.h File

Purpose

The **/usr/include/sys/socket.h** header file provides multiple data definitions and socket structures for use with socket subroutines.

Description

The **/sys/socket.h** file contains data definitions and socket structures required for inclusion in programs using socket subroutines. The **socket.h** file contains data definitions for socket types, address families, and options.

The **socket.h** file also contains the following structures:

- **linger** structure
- **sockaddr** structure
- **sockproto** structure
- **msghdr** structure

linger Structure

The **linger** structure is used for manipulating the linger option. The **linger** structure contains the following members:

l_onoff Toggles the option on or off.

l_linger Specifies the linger time.

sockaddr Structure

The **sockaddr** structure is used by the kernal to store most addresses. The **sockaddr** structure contains the following members:

sa_family Defines the address family of the socket.

sa_data[14] Specifies up to 14 bytes of direct address.

sockproto Structure

The **sockproto** structure is used by the kernal to pass protocol information in raw sockets. The **sockproto** structure contains the following members:

sp_family Specifies the socket address family.

sp_protocol Defines the protocol for the specified address family.

msghdr Structure

The **sendmsg** and **recvmsg** subroutines use the **msghdr** data structure. The **msghdr** structure contains the following members:

msg_name Defines the optional destination address if the socket is unconnected. If no names are needed, use a **NULL** pointer for **msg_name**.

msg_namelen Specifies the size of the **msg_name** address.

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.

Implementation Specifics

The **/usr/include/sys/socket.h** file is part of AIX Base Operating System (BOS) Runtime.

All applications including the **/usr/include/sys/socket.h** file must be compiled with **_BSD** defined. In addition, when applicable, all socket applications must include the BSD library **libbsd**.

Related Information

Socket creation and connection subroutines using the **socket.h** file are the **accept** subroutine, **bind** subroutine, **connect** subroutine, **listen** subroutine, **socket** subroutine, and **socketpair** subroutine.

Status retrieval sockets using the **socket.h** file are the **getsockopt** subroutine, **getpeername** subroutine, and **getsockname** subroutine.

Network address translation subroutines using the **socket.h** file are the **inet_lnaof** subroutine, **inet_addr** subroutine, **inet_makeaddr** subroutine, **inet_network** subroutine, **inet_netof** subroutine, and **inet_ntoa** subroutine.

Data transfer subroutines using the **socket.h** file are the **recv** subroutine, **recvfrom** subroutine, **recvmsg** subroutine, **send** subroutine, **sendto** subroutine, **sendmsg** subroutine, and **shutdown** subroutine.

Header files that contain a **sockaddr** structure are the **/usr/include/netinet/in.h** file and the **/usr/include/sys/un.h** file.

Sockets Overview and Understanding Header Files in *Communications Programming Concepts.*

Sockets socketvar.h File

Purpose

Defines the kernel structure per socket and contains buffer queues.

Description

The **/usr/include/sys/socketvar.h** header file defines the kernel structure per socket for use with socket subroutines. Each application that uses socket subroutines must include the **socketvar.h** file. The **/usr/include/sys/socketvar.h** header file also contains send and receive buffer queues, handle on protocol and pointer to protocol private data and error information.

The fields in the **socketvar.h** file are:

```
 */
struct  socket {
        short    so_type;     /* generic type, see socket.h */
        short    so_options;  /* from socket call, see socket.h */
        short    so_linger;   /* time to linger while closing */
        short    so_state;    /* internal state flags SS_*, below */
        caddr_t  so_pcb;      /* protocol control block */
        struct   protosw *so_proto;     /* protocol handle */

/*
 * Variables for connection queueing.
 * Socket where accepts occur is so_head in all subsidiary sockets.
 * If so_head is 0, socket is not related to an accept.
 * For head socket so_q0 queues partially completed connections,
 * while so_q is a queue of connections ready to be accepted.
 * If a connection is aborted and it has so_head set, then
 * it has to be pulled out of either so_q0 or so_q.
 * We allow connections to queue up based on current queue lengths
 * and limit on number of queued connections for this socket.
 */
 struct    socket *so_head;   /* back pointer to accept socket */
    struct    socket *so_q0;     /* queue of partial connections */
    short     so_q0len;          /* partials on so_q0 */
    struct    socket *so_q;      /* queue of incoming connections */
    short     so_qlen;           /* number of connections on so_q */
    short     so_qlimit;         /* max number queued connections */
```

Variables for Socket Buffering

The variables for socket buffering are as follows:

```
struct  sockbuf {
        u_short   sb_cc;          /* actual chars in buffer */
        u_short   sb_hiwat;       /* max actual char count */
        u_short   sb_mbcnt;       /* chars of mbufs used */
        u_short   sb_mbmax;       /* max chars of mbufs to use */
        u_short   sb_lowat;       /* low water mark (not used) */
        short     sb_timeo;       /* timeout (not used yet) */
        struct    mbuf *sb_mb;    /* the mbuf chain */
        struct    proc *sb_sel;   /* process selecting read/write */
        short     sb_flags;       /* flags, see below */
        int       (*sb_iodone)(); /* I/O done function    */
        caddr_t   sb_ioarg;       /* arg for sb_iodone    */
} so_rcv, so_snd;
#define  SB_MAX       65535  /* max chars in sockbuf */
#define  SB_LOCK       0x01  /* lock on data queue (so_rcv only) */
#define  SB_WANT       0x02  /* someone is waiting to lock */
#define  SB_WAIT       0x04  /* someone is waiting for data/space */
#define  SB_SEL        0x08  /* buffer is selected */
#define  SB_COLL       0x10  /* collision selecting */
#define  SB_KIODONE    0x80  /* kernel I/O done handling    */
        short    so_timeo;   /* connection timeout */
        u_short  so_error;   /* error affecting connection */
        u_short  so_oobmark;   /* chars to oob mark */
        short    so_pgrp;      /* pgrp for signals */
};
```

Socket State Bits

The socket state bits are as follows:

```
#define    SS_NOFDREF          0x001 /* no file table ref any more */
#define    SS_ISCONNECTED       0x002 /* socket connected to a peer */
#define    SS_ISCONNECTING      0x004 /* in process of connecting to
                                        peer */
#define    SS_ISDISCONNECTING   0x008 /* in process of
                                        disconnecting */
#define    SS_CANTSENDMORE      0x010   /* can't send more data to
                                        peer */
#define    SS_CANTRCVMORE       0x020   /* can't receive more data
                                        from peer */
#define    SS_RCVATMARK         0x040   /* at mark on input */

#define    SS_PRIV              0x080  /* privileged for broadcast,
                                        raw... */
#define    SS_NBIO              0x100  /* non-blocking ops */
#define    SS_ASYNC             0x200  /* async i/o notify */
```

Socket Macros

The macros for sockets and socket buffering are as follows:

```
/* how much space is there in a socket buffer (so->so_snd or so->so_
rcv) */
#define    sbspace(sb) \
    (MIN((int)((sb)->sb_hiwat - (sb)->sb_cc),\
     (int)((sb)->sb_mbmax - (sb)->sb_mbcnt)))
```

```
/* do we have to send all at once on a socket? */
#define    sosendallatonce(so) \
    ((so)->so_proto->pr_flags & PR_ATOMIC)

/* can we read something from so? */
#define    soreadable(so) \
    ((so)->so_rcv.sb_cc || ((so)->so_state & SS_CANTRCVMORE) || \
    (so)->so_qlen || (so)->so_error)

/* can we write something to so? */
#define    sowriteable(so) \
    (sbspace(&(so)->so_snd) > 0 && \
    (((so)->so_state&SS_ISCONNECTED) || \
      ((so)->so_proto->pr_flags&PR_CONNREQUIRED)==0) || \
     ((so)->so_state & SS_CANTSENDMORE) || \
     (so)->so_error)

/* adjust counters in sb reflecting allocation of m */
#define    sballoc(sb, m) { \
    (sb)->sb_cc += (m)->m_len; \
    (sb)->sb_mbcnt += MSIZE; \
    if ((m)->m_off > MMAXOFF) \
        (sb)->sb_mbcnt += CLBYTES; \
}

/* adjust counters in sb reflecting freeing of m */
#define    sbfree(sb, m) { \
    (sb)->sb_cc -= (m)->m_len; \
    (sb)->sb_mbcnt -= MSIZE; \
    if ((m)->m_off > MMAXOFF) \
        (sb)->sb_mbcnt -= CLBYTES; \
}

/* set lock on sockbuf sb */
#define sblock(sb) { \
    while ((sb)->sb_flags & SB_LOCK) { \
        (sb)->sb_flags |= SB_WANT; \
        sleep((caddr_t)&(sb)->sb_flags, PZERO+1); \
    } \
    (sb)->sb_flags |= SB_LOCK; \
}

/* release lock on sockbuf sb */
#define    sbunlock(sb) { \
    (sb)->sb_flags &= ~SB_LOCK; \
    if ((sb)->sb_flags & SB_WANT) { \
        (sb)->sb_flags &= ~SB_WANT; \
        wakeup((caddr_t)&(sb)->sb_flags); \
    } \
}

#define    sorwakeup(so)    sowakeup((so), &(so)->so_rcv)
#define    sowwakeup(so)    sowakeup((so), &(so)->so_snd)

#ifdef KERNEL
struct socket *sonewconn();
#endif
```

Implementation Specifics

The **/usr/include/sys/socketvar.h** file is part of AIX Base Operating System (BOS) Runtime.

All applications including the **/usr/include/sys/socketvar.h** file must be compiled with **_BSD** defined. In addition, when applicable, all socket applications must include the BSD library **libbsd**.

Related Information

Header files containing a **sockaddr** type structure are the **/usr/include/sys/socket.h** file **/usr/include/sys/un.h** file and the **/usr/include/netinet/in.h** file.

Sockets Overview and Understanding Header Files in *Communications Programming Concepts*.

Sockets un.h File

Purpose

Defines the structures for the UNIX Interprocess Communication (IPC) domain.

Description

The **/usr/include/sys/un.h** file contains data definitions and a socket structure required for the UNIX IPC domain. An application program must include the **/usr/include/sys/un.h** file for use with socket subroutines and subroutines that specify the UNIX domain.

sockaddr_un Structure

The **sockaddr_un** structure is used to store UNIX domain sockets that require specification of a complete path name. An application program would include the **sockaddr_un** structure for socket communication between local machine processes. The **/usr/include/sys/un.h** file defines the **sockaddr_un** struture as follows:

```
struct  sockaddr_un {

        short   sun_family;     /* AF_UNIX    */
        char    sun_path[108];  /* path name  */
};

#ifdef   _KERNEL
int      unp_discard();
#endif

#endif  /* _H_UN */
```

Implementation Specifics

The **/usr/include/sys/un.h** file is part of AIX Base Operating System (BOS) Runtime.

All applications including the **/usr/include/sys/un.h** file must be compiled with **_BSD** defined. In addition, when applicable, all socket applications must include the BSD library **libbsd**.

Related Information

Additional header files that contain a **sockaddr** structure are the **/usr/include/sys/socket.h** file and the **/usr/include/netinet/in.h** file.

Sockets Overview and Understanding Header Files in *Communications Programming Concepts*.

X.25 x25sdefs.h File

Purpose

Contains the structures used by the X.25 application programming interface (API).

Description

The **/usr/include/x25sdefs.h** file includes the following structures:

Miscellaneous Structures

cb_link_name_struct	Used to indicate the name of the X.25 port.
cb_msg_struct	Used to indicate the type of message being received.
ctr_array_struct	Used to store the counter values and identifiers for use with **x25_ctr_wait**.

Structures Primarily Used for Establishing Calls and Transferring Data

cb_call_struct	Used for calls being made and accepted.
cb_data_struct	Used for the data to be transferred during a call.
cb_fac_struct	Used for information about optional facilities being used.
cb_pvc_alloc_struct	Used to indicate the logical channel number and port assigned to a PVC.

Structures Used for Clearing, Interrupting and Resetting Calls

cb_clear_struct	Used for calls being cleared.
cb_int_data_struct	Used for data sent or received in an interrupt packet.
cb_res_struct	Used for data sent or received in a reset-request packet.

Structures Primarily Used for Managing X.25 Communications:

cb_circuit_info_struct	Used for information about a virtual circuit.
cb_dev_info_struct	Used for information about an X.25 adapter.
cb_link_stats_struct	Used for statistics for an X.25 port.

The structures are listed in alphabetical order.

X.25 cb_call_struct Structure

Used by the **x25_call**, **x25_call_accept**, and **x25_receive** subroutines to pass the name of the X.25 port, called and calling addresses, facilities, and user data.

```
struct cb_call_struct
{
        unsigned long flags;
        char *link_name;
        char *called_addr;
        char *calling_addr;
        struct cb_fac_struct *cb_fac;
        int user_data_len;
        unsigned char *user_data;
} ;
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_D_BIT
: Indicates that the call is going to use D-bit procedures.

X25_FLG_LINK_NAME
: Indicates that the link_name field is used.

X25_FLG_CALLED_ADDR
: Indicates that the called_addr field is used.

X25_FLG_CALLING_ADDR
: Indicates that the calling_addr field is used.

X25_FLG_CB_FAC
: Indicates that the cb_fac field is used.

X25_FLG_USER_DATA
: Indicates that the user_data field is used.

The meanings of the structure fields are shown here:

link_name
: The name of the X.25 port used for an incoming call. Note that this is set to null on received packets.

called_addr
: Pointer to the network user address (NUA) of the called data terminal equipment (DTE). The address is given in ASCIIZ format

calling_addr
: Pointer to the NUA of the calling DTE. The address is given in ASCIIZ format

cb_fac
: Pointer to the facilities information in **cb_fac_struct**.

user_data_len
: Length of field for call user data.

user_data
: Pointer to call user data.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures*.

X.25 Ports and Links Overview
Network User Addresses
Optional X.25 Facilities
X.25 Packet Switching: Making and Receiving a Call
X.25 API: Making and Receiving a Call

X.25 cb_circuit_info_struct Structure

Used by the **x25_circuit_query** subroutine to return information about the circuit.

```
struct cb_circuit_info_struct
{
        unsigned long flags;
        unsigned short lcn;
        unsigned int incoming_packet_size;
        unsigned int outgoing_packet_size;
        unsigned int incoming_throughput_class;
        unsigned int outgoing_throughput_class;
        unsigned int incoming_window_size;
        unsigned int outgoing_window_size;
} ;
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_INCOMING_PACKET_SIZE
: Indicates that the incoming_packet_size field is used.

X25_FLG_OUTGOING_PACKET_SIZE
: Indicates that the outgoing_packet_size field is used.

X25_FLG_INCOMING_THROUGHPUT_CLASS
: Indicates that the incoming_throughput_class field is used.

X25_FLG_OUTGOING_THROUGHPUT_CLASS
: Indicates that the outgoing_throughput_class field is used.

X25_FLG_INCOMING_WINDOW_SIZE
: Indicates that the incoming_window_size field is used.

X25_FLG_OUTGOING_WINDOW_SIZE
: Indicates that the outgoing_window_size field is used.

Fields

The meanings of the structure fields are shown here:

lcn
: Logical channel number

incoming_packet_size
: Actual size for incoming packets

outgoing_packet_size
: Actual size for outgoing packets

incoming_throughput_class
: Throughput class for incoming calls

outgoing_throughput_class
: Throughput class for outgoing calls

incoming_window_size
: Number of incoming packets that can be sent without confirmation

outgoing_window_size
: Number of outgoing packets that can be sent without confirmation.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures*.

Logical Channels and Virtual Circuits
X.25 Packet Attributes

X.25 cb_clear_struct Structure

Used by the **x25_call_clear** and **x25_receive** subroutines to pass the clear cause and diagnostic values, called and calling addresses, facilities information, and user data.

```
struct cb_clear_struct
{
        unsigned long flags;
        u_char cause;
        u_char diagnostic;
        char *called_addr;
        char *calling_addr;
        struct cb_fac_struct *cb_fac;
        int user_data_len;
        u_char *user_data;
};
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_CAUSE
: Indicates that the cause field is used.

X25_FLG_DIAGNOSTIC
: Indicates that the diagnostic field is used.

X25_FLG_CALLED_ADDR
: Indicates that the called_addr field is used.

X25_FLG_CALLING_ADDR
: Indicates that the calling_addr field is used.

X25_FLG_CB_FAC
: Indicates that the cb_fac field is used.

X25_FLG_USER_DATA
: Indicates that the user_data field is used.

Fields

The meanings of the structure fields are shown here:

cause	Cause value to be inserted in clear packet.
diagnostic	Diagnostic reason to be inserted in packet.
called_addr	Pointer to the network user address (NUA) of the called data terminal equipment (DTE). The address is given in ASCIIZ format.
calling_addr	Pointer to the NUA of the calling DTE. The address is given in ASCIIZ format.
cb_fac	Pointer to the facilities information in **cb_fac_struct**.
user_data_len	Length of user-data field.
user_data	Pointer to user data. This can only be used if fast select has been requested in the call-request packet.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures.*

Network User Addresses
Optional X.25 Facilities

X.25 Packet Switching: Clearing, Resetting, and Interrupting
X.25 API: Clearing, Resetting, and Interrupting Calls
List of X.25 Clear and Reset Cause Codes
List of X.25 Diagnostic Codes

X.25 cb_data_struct Structure

Used by the **x25_send** and **x25_receive** subroutines to pass data control information.

```
struct cb_data_struct
{
        unsigned long flags;
        int data_len;
        unsigned char *data;
} ;
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25FLG_D_BIT
: If the D-bit has been set in the call packet, and the value is not zero, the remote data terminal equipment (DTE) *must* acknowledge the packet.

X25FLG_Q_BIT
: Set the Q-bit in the packet. A non-zero value is converted to a single 1-bit in the packet.

X25FLG_M_BIT
: Set the M-bit in the packet. A non-zero value is converted to a single 1-bit in the packet.

X25_FLG_DATA
: Indicates that the data field is used.

Fields

The meanings of the structure fields are shown here:

data_len
: Length of data

data
: Pointer to actual data.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures*.

X.25 Packet Switching: Transferring and Acknowledging Data
X.25 API: Transferring and Acknowledging Data
X.25 Example Program svcxmit: Make a Call Using an SVC
X.25 Example Program svcrcv: Receive a Call Using an SVC
X.25 Example Program pvcxmit: Send Data Using a PVC
X.25 Example Program pvcrcv: Receive Data Using a PVC

X.25 cb_dev_info_struct Structure

Used by the **x25_device_query** subroutine to pass device information.

```
struct cb_dev_info_struct
{
        unsigned long flags;
        char *nua;
        unsigned int no_of_vcs;
        unsigned int max_rx_packet_size;
        unsigned int max_tx_packet_size;
        unsigned int default_svc_rx_packet_size;

        unsigned int default_svc_tx_packet_size;
} ;
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_NUA Indicates that the nua field is used.

X25_FLG_NO_OF_VCS
Indicates that the no_of_vcs field is used.

X25_FLG_MAX_RX_PACKET_SIZE
Indicates that the max_rx_packet_size field is used.

X25_FLG_MAX_TX_PACKET_SIZE
Indicates that the max_tx_packet_size field is used.

X25_FLG_DEFAULT_SVC_RX_PACKET_SIZE
Indicates that the default_svc_rx_packet_size field is used.

X25_FLG_DEFAULT_SVC_TX_PACKET_SIZE
Indicates that the default_svc_tx_packet_size field is used.

Fields

The meanings of the structure fields are shown here:

nua	Pointer to network user address (NUA) recorded for the device in ASCIIZ format.
no_of_vcs	The number of permanent virtual circuits (PVCs) configured on this device.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures.*

Network User Addresses
Logical Channels and Virtual Circuits
List of X.25 Packet Configuration Attributes

X.25 cb_fac_struct Structure

Used by the **x25_call** and **x25_call_accept** subroutines to pass facilities information.

The example shows how to code **cb_fac_struct**.

```
struct cb_fac_struct
{
        u_long flags ;
        unsigned fac_ext_len;
        u_char *fac_ext;        /* for non-X.25 facilities */
        u_char psiz_clg;
        u_char psiz_cld;
        u_char wsiz_clg;
        u_char wsiz_cld;
        u_char tcls_clg;
        u_char tcls_cld;
        unsigned rpoa_id_len;
        ushort *rpoa_id;
        ushort cug_id;
        unsigned nui_data_len;
        u_char *nui_data;
        unsigned ci_seg_cnt_len;
        u_char *ci_seg_cnt;
        unsigned ci_mon_unt_len;
        u_char *ci_mon_unt;
        unsigned ci_cal_dur_len;
        u_char *ci_cal_dur;
        u_char call_redr_addr[X25_MAX_ASCII_ADDRESS_LENGTH];
        u_char call_redr_reason;
        short tran_del;
        u_char calling_addr_ext_use;
        char calling_addr_ext[X25_MAX_EXT_ADDR_DIGITS+1];
        u_char called_addr_ext_use;
        char called_addr_ext[X25_MAX_EXT_ADDR_DIGITS+1];
        u_char clamn;
        u_char min_tcls_clg;
        u_char min_tcls_cld;
        unsigned end_to_end_del_len;
        ushort end_to_end_del[3];
};
```

Several constants are defined in the **x25sdefs.h** file for use by this structure. Following are the descriptions of these constants along with the corresponding fields:

X25FLG_RPOA Recognized private operating agency selection required (rpoa_id).

X25FLG_PSIZ Packet size selection (psiz_clg, psiz_cld).

X25FLG_WSIZ Window size selection (wsiz_clg, wsiz_cld).

X25FLG_TCLS Throughput class required (tcls_clg, tcls_cld).

X25FLG_REV_CHRG
: Reverse Charge required (no corresponding field).

X25FLG_FASTSEL
: Fast select (no corresponding field).

X25FLG_FASTSEL_RSP
: Indicates whether a restricted response is required when X25FLG_FASTSEL is also requested (no corresponding field).

X25FLG_CUG Closed user group selection required (cug_id).

X25FLG_OA_CUG
Closed user group with outgoing access (basic format) selection required (cug_id).

X25FLG_BI_CUG
Bilateral closed user group selection required (cug_id).

X25FLG_NUI_DATA
Network user identification (nui_data).

X25FLG_CI_SEG_CNT
Charging information: segment count (ci_seg_cnt).

X25FLG_CI_MON_UNT
Charging information: monetary unit (ci_mon_unt).

X25FLG_CI_CAL_DUR
Charging information: call duration (ci_cal_dur).

X25FLG_CI_REQUEST
Charging information is required (no corresponding field).

X25FLG_CLAMN
Called line address modified notification (clamn).

X25FLG_CALL_REDR
Call redirection notification (call_redr_addr, call_redr_reason).

X25FLG_TRAN_DEL
Transit delay selection and notification (tran_del).

X25FLG_CALLING_ADDR_EXT
Calling address extension (calling_addr_ext_use, calling_addr_ext).

X25FLG_CALLED_ADDR_EXT
Called address extension (called_addr_ext_use, called_addr_ext).

X25FLG_MIN_TCLS
Quality of service negotiation: minimum throughput class (min_tcls_clg, min_tcls_cld).

X25FLG_END_TO_END_DEL
Quality of service negotiation: end-to-end transit delay (end_to_end_del).

X25FLG_EXP_DATA
Expedited data negotiation (no corresponding field).

X25FLG_FACEXT
Facilities extension: for all other facilities, including national options (fac_ext).

Fields

The meanings of the structure fields are shown here, but the lengths associated with individual pointer fields are not explained:

fac_ext Pointer to the facilities extension array: extra facility information provided by the user or network. No checking is made on the validity of this information. It allows extra facilities that the main **cb_fac** structure does not include. The elements of fac_ext are copied directly into the facility field.

When the information is provided by the X.25 network or by the remote DTE, it is the responsibility of the application to interpret the field.

Only elements up to the first non-X.25 facility are decoded by the API. Facility markers must be used in fac_ext if such facilities are required.

psiz_clg — Indicates the requested size for packets transmitted from the calling DTE. Supported values are:

0x04 = 16 octets
0x05 = 32 octets
0x06 = 64 octets
0x07 = 128 octets
0x08 = 256 octets
0x09 = 512 octets
0x0A = 1024 octets
0x0B = 2048 octets
0x0C = 4096 octets

psiz_cld — Requested size for packets transmitted from the called DTE. Supported values are the same as for psiz_clg.

wsiz_clg — Requested size for the window for packets transmitted by the calling DTE. Values are in the range from 0x01 to 0x07 inclusive.

wsiz_cld — Requested size for the window for packets to be transmitted by the called DTE. Values are in the range from 0x01 to 0x07 inclusive.

tcls_clg — Throughput class requested for data to be sent by the calling DTE. Supported values are:

0x07 = 1200 bits per second
0x08 = 2400 bits per second
0x09 = 4800 bits per second
0x0A = 9600 bits per second
0x0B = 19200 bits per second
0x0C = 48000 bits per second

tcls_cld — Throughput class request for data sent from the called DTE. Supported values are the same as for tcls_clg

rpoa_id — Indicates the requested RPOA transit network. Each element of the array is an RPOA identifier.

cug_id — Indicates the identifier of a closed user group (CUG). Used for all modes of CUG and for bilateral CUGs.

nui_data — Network user identification data in a format specified by the network administrator.

ci_seg_cnt — Charging information: segment count data.

ci_mon_unt — Charging information: monetary unit data.

ci_cal_dur — Charging information: call duration data.

call_redr_addr — The address to which the call has been redirected. The address is stored in ASCIIZ format.

call_redr_reason
: Contains reason for call redirection.

tran_del
: Transit delay in milliseconds.

calling_addr_ext_use
: Indicates the use of the calling address extension.

calling_addr_ext
: Up to 40 digits containing the calling address extension. The address extension is stored in ASCIIZ format. These are the values for the extended calling and called address flags:

X25_FAC_ADDR_EXT_USE_ENTIRE_OSI_NSAP(0)

X25_FAC_ADDR_EXT_USE_PARTIAL_OSI_NSAP(1)

X25_FAC_ADDR_EXT_USE_NON_OSI(2)

called_addr_ext_use
: Indicates the use of the called address extension.

called_addr_ext
: Up to 40 digits containing the called address extension. The address extension is stored in ASCIIZ format. See calling_addr_ext for values.

clamn
: Called line address modified notification. Contains the reason for redirection.

min_tcls_clg
: Throughput class requested for data to be sent by the calling DTE. Supported values are:

  0x07 = 1200 bits per second
  0x08 = 2400 bits per second
  0x09 = 4800 bits per second
  0x0A = 9600 bits per second
  0x0B = 19200 bits per second
  0x0C = 48000 bits per second

min_tcls_cld
: Throughput class request for data sent from the called DTE. Supported values are the same as for min_tcls_clg.

end_to_end_del
: Specifies cumulative, requested end-to-end and maximum-acceptable transit delays. Requested end-to-end and maximum-acceptable values are optional.

Example

This is a simple example of the use of the **cb_fac_struct** structure:

```
/*                                                                    */
struct cb_call_struct cb_call;
struct cb_fac_struct fac_struct;
u_char facilities_extension[10],facilities_extension[8];
ushort rpoa_ext_id[3] = {7,8,9};
char extended_calling_addr[]= "1234567890";  /* Example address
extension */
```

```
/* Initialize flags                                                  */

fac_struct.flags = 0;

/* Use of RPOAE                                                      */

fac_struct.rpoa_id = rpoa_ext_id;
fac_struct.rpoa_id_len = 3;
fac_struct.flags |= X25FLG_RPOA;

/* Use of extended addressing                                        */

fac_struct.calling_addr_ext = extended_calling_addr;
fac_struct.flags |= X25FLG_CALLING_ADDR_EXT;

/* Use of extended facilities                                        */

facilities_extension[0] = 0x00;     /* start of a Facility Marker */
facilities_extension[1] = 0x00;     /* non_X25 facility supported */
                                    /* by calling DTE             */
facilities_extension[2] = 0x55;     /* a facility                 */
facilities_extension[3] = 0x66;     /* a facility                 */
facilities_extension[4] = 0x00;     /* start of a Facility Marker */
facilities_extension[5] = 0xFF;     /* non_X25 facility supported */
                                    /* by called DTE              */
facilities_extension[6] = 0x88;     /* a facility                 */
facilities_extension[7] = 0x99;     /* a facility                 */
strcpy(fac_struct.fac_ext, facilities_extension);
fac_struct.fac_ext_len = 8;
fac_struct.flags |= X25FLG_FACEXT;

/*****************************************************************/
/* In this example a cb_call structure is initialized            */
/* with a cb_fac structure.                                      */
/*****************************************************************/

cb_call.cb_fac = &fac_struct;
cb_call.flags = X25FLG_CB_FAC;
```

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures.*

X.25 Ports and Links Overview
Network User Addresses
Optional X.25 Facilities
X.25 Packet Switching: Making and Receiving a Call
X.25 API: Making and Receiving a Call

X.25 cb_int_data_struct Structure

Used by the **x25_interrupt** and **x25_receive** subroutines to pass the interrupt data.

```
struct cb_int_struct
{
        unsigned long flags ;
        unsigned char int_data_len;
        unsigned char *int_data;
} ;
```

The following constant is defined in the **x25sdefs.h** file for use by this structure:

X25FLG_INT_DATA
: A non-zero value indicates the presence of data in cb_int_data.

Fields

The meanings of the structure fields are shown here:

int_data_len
: Length of data in cb_int_data.

int_data
: Interrupt data.

For additional information related to this structure, refer to the following article in *Communication Concepts and Procedures*.

X.25 Packet Switching: Clearing, Resetting, and Interrupting Calls
X.25 API: Clearing, Resetting, and Interrupting Calls

X.25 cb_link_name_struct Structure

Used by the **x25_init**, **x25_link_connect**, **x25_link_disconnect**, **x25_link_monitor**, **x25_device_query**, and **x25_term** Subroutines to pass the name of the X.25 port.

```
struct cb_link_name_struct
{
        unsigned long flags;
        char *link_name;
};
```

The following constant is defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_LINK_NAME
: Indicates that the link_name field is used.

Fields

The meanings of the structure fields are shown here:

link_name
: The name of the X.25 port.

For additional information related to this structure, refer to the following article in *Communication Concepts and Procedures*.

X.25 Ports and Links Overview

X.25 cb_link_stats_struct, x25_query_data, and x25_stats Structures

Used by the **x25_link_statistics** subroutine to pass statistics about a X.25 port.

```
struct cb_link_stats_struct
{
        unsigned long flags;
        unsigned int no_of_vcs;
        struct x25_query_data x25_stats;
} ;
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_NO_OF_VCS
: Indicates that the no_of_vcs field is used.

X25_FLG_LINK_STATS
: Indicates that the **x25_stats** structure is being used.

Fields

The meanings of the structure fields are shown here:

no_of_vcs — Number of virtual circuits currently in use for the X.25 port specified.

x25_stats — Pointer to **x25_query_data** structure containing CIO and X.25 statistics.

x25_query_data Structure

x25_query_data is the structure returned from the **CIO_QUERY** ioctl. It includes two structures: the standard statistics values found in **sys/comio.h** and the specific X.25 statistics structure, **x25_stats**.

```
struct x25_query_data
{
        struct cio_stats cc;
        struct x25_stats ds;
};
```

Fields

cio_stats — See the **<sys/comio.h>** file.

x25_stats — Pointer to **x25_query_data** structure containing CIO and X.25 statistics.

x25_stats Structure

x25_stats is the structure that contains the specific X.25 statistics. Note that flags are not used with this structure.

```
typedef unsigned short x25_stat_value_t;
struct x25_stats
{
```

Frame Level

```
x25_stat_value_t ignored_f_tx;
x25_stat_value_t rr_f_tx;
x25_stat_value_t rnr_f_tx;
x25_stat_value_t rej_f_tx;
x25_stat_value_t info_f_tx;
x25_stat_value_t sabm_f_tx;
x25_stat_value_t sarm_dm_f_tx;
x25_stat_value_t disc_f_tx;
x25_stat_value_t ua_f_tx;
x25_stat_value_t frmr_f_tx;
x25_stat_value_t bad_nr_f_tx;
x25_stat_value_t unknown_f_tx;
x25_stat_value_t xid_f_tx;
x25_stat_value_t bad_length_f_tx;
x25_stat_value_t t1_expirations;
x25_stat_value_t lvl2_connects;
x25_stat_value_t lvl2_disconnects;
x25_stat_value_t carrier_loss;
x25_stat_value_t connect_time;  /* In seconds */
x25_stat_value_t t4_expirations;
x25_stat_value_t t4_n2_times;
x25_stat_value_t ignored_f_rx;
x25_stat_value_t rr_f_rx;
x25_stat_value_t rnr_f_rx;
x25_stat_value_t rej_f_rx;
x25_stat_value_t info_f_rx;
x25_stat_value_t sabm_f_rx;
x25_stat_value_t sarm_dm_f_rx;
x25_stat_value_t disc_f_rx;
x25_stat_value_t ua_f_rx;
x25_stat_value_t frmr_f_rx;
x25_stat_value_t bad_nr_f_rx;
x25_stat_value_t unknown_f_rx;
x25_stat_value_t xid_f_rx;
x25_stat_value_t bad_length_f_rx;
```

Packet Level

```
x25_stat_value_t data_p_tx;
x25_stat_value_t rr_p_tx;
x25_stat_value_t rnr_p_tx;
x25_stat_value_t interrupt_p_tx;
x25_stat_value_t interrupt_confirm_p_tx;
x25_stat_value_t call_request_p_tx;
x25_stat_value_t call_accept_p_tx;
x25_stat_value_t clear_request_p_tx;
x25_stat_value_t clear_confirm_p_tx;
x25_stat_value_t reset_request_p_tx;
```

```
        x25_stat_value_t reset_confirm_p_tx;
        x25_stat_value_t diagnostic_p_tx;
        x25_stat_value_t registration_p_tx;
        x25_stat_value_t registration_confirm_p_tx;
        x25_stat_value_t restart_p_tx;
        x25_stat_value_t restart_confirm_p_tx;
        x25_stat_value_t error_p_tx;
        x25_stat_value_t t20_expirations;
        x25_stat_value_t t21_expirations;
        x25_stat_value_t t22_expirations;
        x25_stat_value_t t23_expirations;
        x25_stat_value_t vc_establishments;
        x25_stat_value_t t24_expirations;
        x25_stat_value_t t25_expirations;
        x25_stat_value_t t26_expirations;
        x25_stat_value_t t28_expirations;
        x25_stat_value_t data_p_rx;
        x25_stat_value_t rr_p_rx;
        x25_stat_value_t rnr_p_rx;
        x25_stat_value_t interrupt_p_rx;
        x25_stat_value_t interrupt_confirm_p_rx;
        x25_stat_value_t incoming_call_p_rx;
        x25_stat_value_t call_connected_p_rx;
        x25_stat_value_t clear_indication_p_rx;
        x25_stat_value_t clear_confirm_p_rx;
        x25_stat_value_t reset_indication_p_rx;
        x25_stat_value_t reset_confirm_p_rx;
        x25_stat_value_t diagnostic_p_rx;
        x25_stat_value_t registration_p_rx;
        x25_stat_value_t registration_confirm_p_rx;
        x25_stat_value_t restart_p_rx;
        x25_stat_value_t restart_confirm_p_rx;
        int transmit_profile [16];
        int receive_profile [16];
};
```

Fields

The meanings of the structure fields are shown here:

ignored_f_tx	Count of the number of transmitted frames that have been ignored instead of being transmitted.
rr_f_tx	Count of the number of RR frames transmitted.
rnr_f_tx	Count of the number of RNR frames transmitted.
rej_f_tx	Count of the number of REJ frames transmitted.
info_f_tx	Count of the number of INFO frames transmitted.
sabm_f_tx	Count of the number of SABM frames transmitted.
sarm_dm_f_tx	Count of the number of SARM/DM frames transmitted.
disc_f_tx	Count of the number of DISC frames transmitted.
ua_f_tx	Count of the number of UA frames transmitted.

frmr_f_tx Count of the number of FRMR frames transmitted.

bad_nr_f_tx Count of the number of frames transmitted with a bad N(R) value.

unknown_f_tx Count of the number of unknown frames transmitted.

xid_f_tx Count of the number of XID frames transmitted.

bad_length_f_tx Count of the number of bad length frames transmitted.

t1_expirations Count of the number of times the T1 timer has timed out.

lvl2_connects Count of the number of times the frame level has been connected.

lvl2_disconnects
Count of the number of times the frame level has been disconnected.

carrier_loss Count of the number of times the carrier signal was lost.

connect_time The number of seconds that the link has been connected.

t4_expirations Count of the number of times the T4 timer has timed out.

t4_n2_expirations
Count of the number of times the T4 timer has timed out N2 times.

ignored_f_rx Count of the number of received frames that have been ignored instead of being received.

rr_f_rx Count of the number of RR frames received.

rnr_f_rx Count of the number of RNR frames received.

rej_f_rx Count of the number of REJ frames received.

info_f_rx Count of the number of INFO frames received.

sabm_f_rx Count of the number of SABM frames received.

sarm_dm_f_rx Count of the number of SARM/DM frames received.

disc_f_rx Count of the number of DISC frames received.

ua_f_rx Count of the number of UA frames received.

frmr_f_rx Count of the number of FRMR frames received.

bad_nr_f_rx Count of the number of frames received with a bad N(R) value.

unknown_f_rx Count of the number of unknown frames received.

xid_f_rx Count of the number of XID frames received.

bad_length_f_rx
Count of the number of bad length frames received.

data_p_tx Count of the number of data packets transmitted.

rr_p_tx Count of the number of RR packets transmitted.

rnr_p_tx Count of the number of RNR packets transmitted.

interrupt_p_tx Count of the number of interrupt packets transmitted.

interrupt_confirm_p_tx
Count of the number of interrupt-confirmation packets transmitted.

call-request_p_tx
Count of the number of call-request packets transmitted.

call_accept_p_tx
Count of the number of call-accept packets transmitted.

clear_request_p_tx
Count of the number of clear-request packets transmitted.

clear_confirm_p_tx
Count of the number of clear-confirm packets transmitted.

reset_request_p_tx
Count of the number of reset-request packets transmitted.

reset_confirm_p_tx
Count of the number of reset-confirm packets transmitted.

diagnostic_p_tx Count of the number of diagnostic packets transmitted.

registration_p_tx
Count of the number of registration packets transmitted.

registration_confirm_p_tx
Count of the number of registration-confirmation packets transmitted.

restart_p_tx Count of the number of restart packets transmitted.

restart_confirm_p_tx
Count of the number of restart-confirmation packets transmitted.

error_p_tx Count of the number of error packets transmitted.

t20_expirations Count of the number of times the T20 timer has timed out.

t21_expirations Count of the number of times the T21 timer has timed out.

t22_expirations Count of the number of times the T22 timer has timed out.

t23_expirations Count of the number of times the T23 timer has timed out.

vc_establishments
Count of the number of times a virtual circuit has been established.

t24_expirations Count of the number of times the T24 timer has timed out.

t25_expirations Count of the number of times the T25 timer has timed out.

t26_expirations Count of the number of times the T26 timer has timed out.

t28_expirations Count of the number of times the T28 timer has timed out.

data_p_rx Count of the number of data packets received.

rr_p_rx Count of the number of RR packets received.

rnr_p_rx Count of the number of RNR packets received.

interrupt_p_rx Count of the number of interrupt packets received.

interrupt_confirm_p_rx
Count of the number of interrupt-confirmation packets received.

call-request_p_rx
Count of the number of call-request packets received.

call_accept_p_rx
Count of the number of call-accept packets received.

clear_request_p_rx
Count of the number of clear-request packets received.

clear_confirm_p_rx
Count of the number of clear-confirm packets received.

reset_request_p_rx
Count of the number of reset-request packets received.

reset_confirm_p_rx
Count of the number of reset-confirm packets received.

diagnostic_p_rx Count of the number of diagnostic packets received.

registration_p_rx
Count of the number of registration packets received.

registration_confirm_p_rx
Count of the number of registration-confirmation packets received.

restart_p_rx Count of the number of restart packets received.

restart_confirm_p_rx
Count of the number of restart-confirmation packets received.

transmit_profile[16]
A profile of the transmission packet sizes in use on this X.25 port. Each element of the array contains a count of the number of packets (sent since the X.25 adapter was last configured) whose sizes are in the range specified (as shown below).

0	Packet size not known.
1	Reserved.
2	Reserved.
3	Reserved.
4	0 through 15.
5	16 through 31.
6	32 through 63.

7	64 through 127.
8	128 through 255.
9	256 through 511.
10	512 through 1023.
11	1024 through 2047.
12	2048 through 4095.
13 — 16	Reserved.

receive_profile[16]
A profile of the receive packet sizes in use on this X.25 port. Each element of the array contains a count of the number of packets (received since the X.25 adapter was last configured) whose sizes are in the range specified (as for transmit_profile).

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures*.

X.25 Ports and Links Overview
Logical Channels and Virtual Circuits
Example of X.25 Statistics
How to Get Statistics For an X.25 Port

X.25 cb_msg_struct Structure

Used by the **x25_receive** and **x25_call_clear** subroutines to pass the contents of a received packet to an application.

```
struct cb_msg_struct
{
        int msg_type;
        union
        {
                struct cb_call_struct *cb_call;
                struct cb_data_struct *cb_data;
                struct cb_clear_struct *cb_clear;
                struct cb_res_struct *cb_res;
                struct cb_int_struct *int_data;
        } msg_point;
};
```

Fields

The meanings of the structure fields are shown here:

msg_type Type of message being returned, as follows:

X25_CALL_CONNECTED
Call connected: cb_call points to the **cb_call_struct** structure.

X25_INCOMING_CALL
Incoming call: cb_call points to the **cb_call_struct** structure.

X25_DATA Data: cb_data points to the **cb_data_struct** structure.

X25_DATA_ACK
Data acknowledgement: no buffer.

X25_INTERRUPT
Interrupt: int_data points to the **cb_int_data_struct** structure.

X25_INTERRUPT_CONFIRMATION
: Confirmation of a previously issued interrupt request: no data is returned.

X25_CLEAR_INDICATION
: Indication that call has been cleared.

X25_CLEAR_CONFIRM
: Confirmation that the call has been cleared. cb_clear points to the **cb_clear_struct** structure. (This should only be received on **x25_call_clear**.)

X25_RESET_INDICATION
: Reset indication: cb_res points to the **cb_res_struct** structure.

X25_RESET_CONFIRM
: Reset confirmation: no data is returned.

X25_UNKNOWN_PACKET
: Allow for packets in future CCITT releases. These packets will be ones that can be safely ignored by the application.

cb_call — Pointer to the call structure, **cb_call_struct**.

cb_data — Pointer to the data structure, **cb_data_struct**.

cb_clear — Pointer to the clear structure, **cb_clear_struct**.

cb_res — Pointer to the reset structure, **cb_res_struct**.

int_data — Pointer to the interrupt data structure, **cb_int_data_struct**.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures*.

X.25 API: Making and Receiving a Call
X.25 API: Transferring and Acknowledging Data
X.25 API: Clearing, Resetting, and Interrupting Calls
X.25 Example Program svcxmit: Make a Call Using an SVC
X.25 Example Program svcrcv: Receive a Call Using an SVC

X.25 cb_pvc_alloc_struct Structure

Used by the **x25_pvc_alloc** subroutine to pass the name of the X.25 port and the logical channel number.

```
struct cb_pvc_alloc_struct
{
        unsigned long flags;
        char *link_name;
        unsigned int lcn;
} ;
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_LCN Indicates that the lcn field is used.

X25_FLG_LINK_NAME
: Indicates that the link_name field is used.

Fields

The meanings of the structure fields are shown here.

link_name — The name of the X.25 port.

lcn — Logical channel number of the permanent virtual circuit (PVC) to be allocated to the call.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures*.

X.25 Ports and Links Overview
Logical Channels and Virtual Circuits
X.25 Example Program pvcxmit: Send Data Using a PVC
X.25 Example Program pvcrcv: Receive Data Using a PVC

X.25 cb_res_struct Structure

Used by the **x25_reset** and **x25_receive** subroutines to pass the reset cause and diagnostic codes.

```
struct cb_res_struct
{
        unsigned long flags;
        unsigned char cause;
        unsigned char diagnostic;
};
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_CAUSE
Indicates that the cause field is used.

X25_FLG_DIAGNOSTIC
Indicates that the diagnostic field is used.

Fields

The meanings of the structure fields are shown here.

cause — Cause value of either 0 or in the range 0x80 through 0xFF, to be inserted in the reset packet.

diagnostic — Diagnostic reason to be inserted in the packet. The CCITT default value is 0.

For additional information related to this structure, refer to the following articles in *Communication Concepts and Procedures*.

X.25 Packet Switching: Clearing, Resetting, and Interrupting Calls
X.25 API: Clearing, Resetting, and Interrupting Calls
List of X.25 Clear and Reset Cause Codes
List of X.25 Diagnostic Codes

X.25 ctr_array_struct Structure

Used by the **x25_ctr_wait** subroutine to pass the counter identifier and a value to be exceeded.

```
struct ctr_array_struct
{
        unsigned long flags;
        int ctr_id;
        int ctr_value;
} ;
```

The following constants are defined in the **x25sdefs.h** file for use by this structure:

X25_FLG_CTR_ID
: Indicates that the ctr_id field is used.

X25_FLG_CTR_VALUE
: Indicates that the ctr_value field is used.

Fields

The meanings of the structure fields are shown here:

ctr_id
: Counter identifier.

ctr_value
: Value to be exceeded by the counter specified by the counter identifier. The counter is incremented each time a message for the associated call or PVC arrives, so **x25_ctr_wait** returns control to the calling program when this number of messages are waiting.

For additional information related to this structure, refer to the following article in *Communication Concepts and Procedures.*

X.25 API: Using Counters to Correlate Messages

Files

/usr/include/x25sdefs.h The path to the **x25sdefs.h** header file.

Related Information

Using X.25 Applications Written for Previous Releases in *Communication Concepts and Procedures.*
Using the X.25 Subroutines in *Communication Concepts and Procedures.*
Using the X.25 Structures and Flags in *Communication Concepts and Procedures.*

Chapter 5. Directories

BNU /etc/locks Directory

Purpose

Contains lock files that prevent multiple uses of communications devices and multiple calls to remote systems.

Description

The **/etc/locks** directory contains files that lock communications devices and remote systems so that another user cannot access them when they are already in use. Other programs check the **/etc/locks** directory for lock files before attempting to use a particular device or call a specific system.

A lock file is a file that is placed in the **/etc/locks** directory when a program uses a communications device or contacts a remote system. The file contains the process ID number (PID) of the process that creates it.

The Basic Networking Utilities Program and other communications programs create a device lock file whenever a connection to a remote system, established over the specified device, is actually in use. The full path name of a device lock file is:

/etc/locks/*DeviceName*

where the *DeviceName* extension is the name of a device, such as `tty3`.

When the Basic Networking Utilities (BNU) **uucico** daemon, **cu** command, or **tip** command place a call to a remote system, they put a system lock file in the **/etc/locks** directory. The full path name of a system lock file is:

/etc/locks/*SystemName*

where the *SystemName* extension is the name of a remote system, such as `hera`. The system lock file prevents more than one connection at a time to the same remote system.

Under normal circumstances, the communications software automatically removes the lock file when the user or program ends the connection to a remote system. However, if a process executing on the specified device or system does not complete its run (for example, if the computer crashes), the lock file remains in the **/etc/locks** directory either until the file is removed manually or until the system is restarted after a shutdown.

Implementation Specifics

This directory is part of AIX Base Operating System (BOS) Runtime.

Files

/etc/locks/*DeviceName*	Prevents multiple uses of the device named by the *DeviceName* file.
/etc/locks/*SystemName*	Prevents multiple connections to the remote system named by the *SystemName* file.

Related Information

The following commands and daemons place lock files in the **/etc/locks** directory: the **connect** subcommand of the ATE command, **pshare** command, **pdelay** command, **uucico** daemon, **ct** command, **cu** command, **slattach** command, **tip** command.

BNU /usr/spool/uucp Directory

Purpose

Stores Basic Networking Utilities (BNU) log, administrative, command, data, and execute files in multiple subdirectories.

Description

The **/usr/spool/uucp** directory, also known as the BNU spooling directory, is the parent directory for multiple work directories created by the Basic Networking Utilities (BNU) program to facilitate file transfers among systems.

The following directories are subdirectories of the **/usr/spool/uucp** directory:

.Admin Contains four administrative files. They are as follows:

audit

- **Foreign**
- **errors**
- **xferstats**

.Corrupt Contains copies of files that could not be processed by the BNU program.

.Log Contains log files for the **uucico** and **uuxqt** daemons.

.Old Contains old log files for the **uucico** and **uuxqt** daemons.

.Status Stores the last time the **uucico** daemon tried to contact remote systems.

.Workspace Holds temporary files that the file transport programs use internally.

.Xqtdir Contains execute files with lists of commands that remote systems can run.

SystemName Contains files used by file transport programs, including:

- Command (**C.***) files
- Data (**D.***) files
- Execute (**X.***) files
- Temporary (**TM.***) files.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **uucico** daemon, **uuxqt** daemon.

The **uucp** command, **uudemon.cleanu** command, **uuclean** command, **uupick** command, **uuto** command, **uux** command.

BNU /usr/spool/uucp/.Admin Directory

Purpose

Contains administrative files used by BNU.

Description

The **/usr/spool/uucp/.Admin** directory contains administrative files used by the Basic Networking Utilities (BNU) program to facilitate remote communications among systems. The **.Admin** directory contains the following files:

File	Description
audit	Contains debug messages from the **uucico** daemon.
Foreign	Logs contact attempts from unknown remote systems.
errors	Records **uucico** daemon errors.
xferstats	Records the status of file transfers.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **cron** daemon, **uucico** daemon.

The **uudemon.cleanu** command.

Description

The **/usr/spool/uucp/.Admin** directory contains administrative files used by the Basic Networking Utilities (BNU) program to facilitate remote communications among systems. The **.Admin** directory contains the following files:

File	Description
audit	Contains debug messages from the **uucico** daemon.
Foreign	Logs contact attempts from unknown remote systems.
errors	Records **uucico** daemon errors.
xferstats	Records the status of file transfers.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **cron** daemon, **uucico** daemon.

The **uudemon.cleanu** command.

BNU /usr/spool/uucp/.Corrupt Directory

Purpose

Contains copies of files that could not be processed.

Description

The **/usr/spool/uucp/.Corrupt** directory contains copies of files that could not be processed by the Basic Network Utilities (BNU) program. For example, if a file is not in the correct form for transfer, the BNU program places a copy of that file in the **.Corrupt** directory for later handling. This directory is rarely used.

The files in the **.Corrupt** directory are removed periodically by the **uudemon.cleanu** command, a shell procedure.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **uucico** daemon, **uuxqt** daemon.

The **uudemon.cleanu** command.

BNU /usr/spool/uucp/.Log Directories

Purpose

Contain the BNU program log files.

Description

The **/usr/spool/uucp/.Log** directories contain Basic Networking Utilities (BNU) program log files. The BNU program normally places status information about each transaction in the appropriate log file each time you use the networking utilities facility.

All transactions of the **uucico** and **uuxqt** daemons as well as the **uux** and **uucp** commands are logged-in files named for the remote system concerned. Each file is stored in a subdirectory of the **/usr/spool/uucp/.Log** directory, named for the daemon or command involved. Each subdirectory contains a separate file for each remote system contacted. Thus the log files are named with a form of the following:

/usr/spool/uucp/.Log/*DaemonName*/*SystemName*

OR

/usr/spool/uucp/.Log/*CommandName*/*SystemName*

All activities of the **uucp** command are logged in the *SystemName* file in the **/usr/spool/uucp/.Log/uucp** directory. All activities of the **uux** command are logged in the *SystemName* file in the **/usr/spool/uucp/.Log/uux** directory.

The **uucp** and **uuto** commands call the **uucico** daemon. The **uucico** daemon activities for a particular remote system are logged in the *SystemName* file in the **/usr/spool/uucp/.Log/uucico** directory on the local system.

The **uux** command calls the **uuxqt** daemon. The **uuxqt** daemon activities for a particular remote system are logged in the *SystemName* file in the **/usr/spool/uucp/.Log/uuxqt** directory on the local system.

When more than one BNU process is running, however, the system cannot access the standard log file, so it places the status information in a file with a **.Log** prefix. The file covers that single transaction.

The BNU program can automatically append the temporary log files to a primary log file. This is called *compacting the log files* and is handled by the **uudemon.cleanu** command, a shell procedure. The procedure combines the log files of the activities of the **uucico** and **uuxqt** daemons on a particular system and stores the files in the **/usr/spool/uucp/.Old** directory.

The default is for the **uudemon.cleanu** command to save log files that are two days old. This default can be changed by modifying the appropriate line in the shell procedure. If storage space is a problem on a particular system, reduce the number of days that the files are kept in their individual log files.

The **uulog** command can be used to view the BNU program log files.

Implementation Specifics

These directories are part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

Working with BNU Log Files in *Communication Concepts and Procedures*.

The **cron** daemon, **uucico** daemon, **uusched** daemon, **uuxqt** daemon.

The **uucp** command, **uudemon.cleanu** command, **uulog** command, **uuto** command, **uux** command.

BNU /usr/spool/uucp/.Old Directory

Purpose

Contains the combined BNU program log files.

Description

The **/usr/spool/uucp/.Old** directory contains the combined Basic Networking Utilities (BNU) program log files.

The BNU program creates log files of the activities of the **uucico** and **uuxqt** daemons in the **/usr/spool/uucp/.Log** directory. The log files are compacted by the **/usr/lib/uucp/uudemon.cleanu** command, a shell procedure, which combines the files and stores them in the **.Old** directory.

By default, the **uudemon.cleanu** command removes log files after two weeks. The length of time log files are kept can be changed to suit the needs of an individual system.

The log files can be viewed using the **uulog** command.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

Working with BNU Log Files in *Communication Concepts and Procedures*.

The **cron** daemon, **uucico** daemon, **uuxqt** daemon.

The **uucp** command, **uudemon.cleanu** command, **uulog** command, **uux** command.

BNU /usr/spool/uucp/.Status Directory

Purpose

Contains information about the status of the BNU program contacts with remote systems.

Description

The **/usr/spool/uucp/.Status** directory contains information about the status of the Basic Networking Utilities (BNU) program contacts with remote systems.

For each remote system contacted, the BNU program creates a file in the **.Status** directory called *SystemName*, which is named for the remote system being contacted. In the **.Status**/*SystemName* file, the BNU program stores the following information:

- Time of the last call in seconds
- Status of the last call
- Number of retries
- Retry time, in seconds, of the next call.

Note: The times given in the **.Status**/*SystemName* file are expressed as seconds elapsed since midnight of January 1, 1970 (the output of a **time** subroutine). Thus the retry time is in the form of the number of seconds that must have expired since midnight of January 1, 1970 before the system can retry. To make this entry in the **.Status**/*SystemName* file, BNU performs a **time** subroutine, adds 600 seconds, and places the resulting number of seconds in the file.

If the last call was unsuccessful, the **uucico** daemon will wait until the time specified by the retry time before attempting to contact the system again. The retry time in the **.Status**/*SystemName* file can be overridden using the **–r** flag of the **uutry** or **Uutry** command.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **uucico** daemon.

The **uutry** command, **Uutry** command.

The **time** subroutine.

BNU /usr/spool/uucp/*SystemName* Directories

Purpose

Contain queued requests for file transfers and command executions on remote systems.

Description

The **/usr/spool/uucp/***SystemName* directories are the Basic Networking Utilities (BNU) spooling directories on the local system. The BNU program creates a *SystemName* directory for each system listed in the **/usr/lib/uucp/Systems** file, including the local system.

Each *SystemName* directory contains queued requests issued by local users for file transfers to remote systems and for command executions on remote systems.

The BNU program uses several types of administrative files to transfer data between systems. These files are stored in the *SystemName* directories. They are:

File	Description
command (**C.***) files	Contain directions for the **uucico** daemon concerning file transfers.
data (**D.***) files	Contain data to be sent to remote systems by the **uucico** daemon.
execute (**X.***) files	Contain instructions for running commands on remote systems.
temporary (**TM.***) files	Contain data files after their transfer to the remote system until the BNU program can deliver them to their final destinations (usually the **/usr/spool/uucppublic** directory).

Implementation Specifics

These directories are part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **uucico** daemon, **uusched** daemon, **uuxqt** daemon.

The **uucp** command, **uux** command.

BNU /usr/spool/uucp/.Workspace Directory

Purpose

Holds temporary files used internally by file transport programs.

Description

The **/usr/spool/uucp/.Workspace** directory holds temporary files of various kinds used internally by BNU file transport programs.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **uucico** daemon, **uuxqt** daemon.

BNU /usr/spool/uucp/.Xqtdir Directory

Purpose

Contains temporary files used by the **uuxqt** daemon to execute remote command requests.

Description

The **/usr/spool/uucp/.Xqtdir** directory contains temporary files used by the Basic Networking Utilities (BNU) **uuxqt** daemon to execute remote command requests.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

The **uuxqt** daemon.

The **uux** command.

BNU /usr/spool/uucppublic Directory

Purpose

Stores BNU files until they can be delivered.

Description

The **/usr/spool/uucppublic** directory is the public directory for the Basic Networking Utilities (BNU) facility. One of these directories exists on every system connected by the BNU utilities.

When a user transfers a file to a remote system or issues a request to execute a command on another system, the files generated by these BNU commands are stored in the public directory on the designated system until the destination directory is ready to receive them. (A user can also specify a destination other than the public directory when issuing the **uucp**, **uuto**, or **uux** command.) The transferred files remain in the **uucppublic** directory until they are removed manually or automatically.

Note: The files are stored in the **uucppublic**/*SystemName* subdirectory of the **uucppublic** directory, where the *SystemName* directory is named for the remote system where the files originated.

All spooling directories are dynamic, including the public directory. Depending upon the size of your installation and the number of files sent to the local **/usr/spool/uucppublic** directory by users on remote systems, this directory can at times become quite large.

The **uudemon.cleanu** command, a shell procedure, cleans up all the BNU spooling directories, including the public directories. Use the **uucleanup** command with the **-s***SystemName* flag to clean up the directories on one specific system.

Implementation Specifics

This directory is part of the UNIX to UNIX Copy Program in BOS Extensions 1 of AIX for RISC System/6000.

Related Information

Commands used to clean up the spooling directories are the **uucleanup** command and **uudemon.cleanu** command.

Commands used to make transfer and command requests are the **uucp** command, **uuto** command, and **uux** command.

HCON /usr/lib/hcon Directory

Purpose

Contains files used by the AIX 3270 Host Connection Program/6000 (HCON).

Description

The **/usr/lib/hcon** directory contains files used by the AIX 3270 Host Connection Program/6000 (HCON). It contains color and keyboard definition files, terminal definition files, HCON API subdirectories, AUTOLOG and LAF object and executable files, AUTOLOG and LAF example files, configuration data base files, and the command to start the HCON subsystem.

Color and Keyboard Definition Files

The following files contain data used to define and customize the HCON color and keyboard definition tables:

File	Contents
e789_ctbl	Default binary color definition table
e789_ctbl.p	Source for the default binary color definition table
e789_ktbl	Default binary keyboard definition table
e789_ktbl.p	Source for the default binary keyboard definition table
func_names	Keyboard function names
keynames	Key names
nls_names	Color and attribute names.

The color and keyboard definition tables in the **/usr/lib/hcon** directory specify defaults for use by HCON emulator sessions. The **e789cdef** and **e789kdef** commands allow users to customize color and keyboard definition tables.

The **func_names**, **keynames**, and **nls_names** files define the names used for 3270 functions, keys, colors, and attributes. These names can be changed by editing the files with an ASCII text editor. The names in these files can be in any language and are used as input to the **e789kdef** and **e789cdef** commands. Changes to these files affect the entire HCON color and keyboard customization process.

Terminal Definition Files

The HCON installation process creates a **terminfo** subdirectory in the **/usr/lib/hcon** directory. The **/usr/lib/hcon/terminfo** directory contains terminal definition files that are specific to HCON. When HCON is installed, the **terminfo** directory contains the following files:

File	Contents
ibm.ti.H	Terminal definitions for HFT, IBM 5081, IBM 3151, IBM 3161, IBM 3162, and IBM 3163 terminals.
dec.ti.H	Terminal definitions for DEC VT100 and DEC VT220 terminals.
wyse.ti.H	Terminal definition for the WYSE WY–50 terminal.

The **terminfo** binary files for HCON terminal definitions are in subdirectories of the **/usr/lib/hcon/terminfo** directory. Each subdirectory is named with the first letter of the terminal name. When HCON is installed, the **terminfo** directory contains the following subdirectories:

Subdirectory	Contents
a	Binary terminal definition file for running within AIXwindows
h	Binary terminal definition files for color and monochrome HFTs
i	Binary terminal definition files for the IBM 5081, IBM 3151, IBM 3161, IBM 3162, and IBM 3163 terminals
v	Binary terminal definition files for the DEC VT100 and DEC VT220 terminals
w	Binary terminal definition files for the WYSE WY–50 terminal.

In addition to those delivered with HCON, the **/usr/lib/hcon/terminfo** subdirectory can contain customized terminal definitions.

HCON API Subdirectories

The HCON installation process creates two subdirectories in the **/usr/lib/hcon** directory that contain files used by the HCON API:

Directory	Contents
mvs	API programs to use in interfaces to MVS/TSO host systems, including the **instalapi** program
vm	API programs to use in interfaces to VM/CMS host systems, including the **instalapi** program.

AUTOLOG and LAF Object and Executable Files

When HCON is installed, the **/usr/lib/hcon** directory contains the following AUTOLOG and LAF object and executable files:

File	Contents
autolog.o	AUTOLOG object file
dfxfer.o	File-transfer process object file
laf.o	LAF object file
tlaf.o	LAF test-tool object file
tlaf	Pre-compiled AUTOLOG test-tool executable file.

AUTOLOG and LAF Example Files

The **/usr/lib/hcon** directory contains several example files for the AUTOLOG and LAF facilities. These files are:

File	Contents
g_log.mvs	Example LAF program for MVS/TSO host
g_log.vm	Example LAF program for VM/CMS host
logform	Example **genprof** form for creating AUTOLOG procedures

SYStso	Example AUTOLOG script for MVS/TSO host
SYSvm1	Example AUTOLOG script for VM/CMS host
SYSvm2	Example AUTOLOG script for VM/CMS host.

Configuration Data Base Files

The following files contain HCON configuration information. This information is used by HCON programs, by the Object Data Manager (ODM), and by the HCON configuration commands, which are called by the System Management Interface Tool (SMIT).

File	Contents
sysdflts	HCON database system defaults
sysdflts.vc	HCON database system defaults
users	HCON users database.

Command to Start the HCON Subsystem

The **sthcondmn** command is used to start the **hcondmn** subsystem after HCON has been installed.

Implementation Specifics

This directory is part of the AIX 3270 Host Connection Program/6000 (HCON).

Related Information

Customizing HCON in *Communication Concepts and Procedures* discusses the options available to you for customizing HCON color and keyboard definitions, key and color names, and terminal definitions.

Understanding HCON Programming Examples in *Communications Programming Concepts* discusses the API directories.

Understanding AUTOLOG and Understanding the Logon Assist Feature (LAF) in *Communications Programming Concepts* discuss the AUTOLOG and LAF facilities.

The **e789cdef** command and **e789kdef** command create customized color and keyboard definition tables.

Mail /usr/spool/mqueue Directory

Purpose

Contains the **log** file and temporary files associated with the messages in the mail queue.

Description

The **/usr/spool/mqueue** directory contains temporary files associated with the messages in the mail queue and may contain the **log** file. For further information see the **syslogd** daemon.

Temporary files have names that include the mail queue ID (*MQID*) of the message for which the file was created:

df*MQID*	Data file
lf*MQID*	Lock file
nf*MQID*	Backup file
qf*MQID*	Queue control file
tf*MQID*	Temporary control file
xf*MQID*	Transcript file for session.

Implementation Specifics

This directory is part of AIX Base Operating System (BOS) Runtime.

File

/usr/spool/mqueue directory	Contains the **log** file and temporary files associated with the messages in the mail queue.

Related Information

The **sendmail** command.

The **syslogd** daemon.

How to Examine the Message Queue Files in *Communication Concepts and Procedures.*

Understanding the Mail Queue in *Communication Concepts and Procedures.*

Index

Numbers

A

B

C

D

E

F

G

H

N

O

P

Q

R

S

T

U

V

W

X

Reader's Comment Form

AIX Files Reference for IBM RISC System/6000

SC23–2200–00

Please use this form only to identify publication errors or to request changes in publications. Your comments assist us in improving our publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, 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.

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page	Comments

Please contact your IBM representative or your IBM–approved remarketer to request additional publications.

Please print

Date _____

Your Name _____

Company Name _____

Mailing Address _____

_____

_____

Phone No. ( ) _____
Area Code

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 997
11400 Burnet Rd.
Austin, Texas 78758–3493

Fold

Fold

Cut or Fold Along Line

Fold and Tape

Please Do Not Staple

Fold and Tape

International Business Machines
Corporation
11400 Burnet Road
Austin, Texas 78758–3493

Printed in the
United States of America

SC23-2200-00