Bull

AIX 4.3 Files Reference

AIX

ORDER REFERENCE
86 A2 79AP 06

Bull

AIX 4.3 Files Reference

AIX

Software

April 2000

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 79AP 06

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

AIX® is a registered trademark of International Business Machines Corporation, and is being used under licence.

UNIX is a registered trademark in the United States of America and other countries licensed exclusively through the Open Group.

Year 2000

The product documented in this manual is Year 2000 Ready.

Table of Contents

First Edition (October 1997)

This edition of *AIX Version 4.3 Files Reference* applies to the AIX Version 4, 3270 Host Connection Program 2.1 and 1.3.3 for AIX, and Distributed SMIT 2.2 for AIX licensed programs, and to all subsequent releases of these products until otherwise indicated in new releases or technical newsletters.

A reader’s comment form is provided at the back of this publication. If the form has been removed, address comments to Publications Department, Internal Zip 9561, 11400 Burnet Road, Austin, Texas 78758-3493. To send comments electronically, use this commercial internet address: aix6kpub@austin.ibm.com. Any information that you supply may be used without incurring any obligation to you.

Trademarks and Acknowledgements

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

Tektronix is a trademark of Tektronix, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

VAX is a trademark of Digital Equipment Corporation.

WYSE is a trademark of WYSE Corporation.

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

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

About This Book

This book, *AIX Version 4.3 Files Reference*, describes the files used by the Advanced Interactive Executive Operating System (AIX). 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 experienced C programmers. To use the book effectively, you should be familiar with AIX or UNIX System V commands, system calls, subroutines, file formats, and special files.

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. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
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.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related Publications

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

- *AIX and Related Products Documentation Overview*, Order Number SC23-2456.
- *AIX Version 4.3 Quick Reference*, Order Number SC23-2529.
- *AIX Version 4.3 System User's Guide: Operating System and Devices*, Order Number SC23-4121.
- *AIX Version 4.3 System User's Guide: Communications and Networks*, Order Number SC23-4122
- *AIX Version 4.3 System Management Guide: Operating System and Devices*, Order Number SC23-4126.
- *AIX Version 4.3 System Management Guide: Communications and Networks*, Order Number SC23-4127.
- *AIX Version 4.3 Commands Reference*, Order Number SBOF-1877 (six volumes).
- *AIX Version 4.3 Technical Reference*, Order Number SBOF-1878 (six volumes).
- *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*, Order Number SC23-4128.
- *AIX Version 4.3 Communications Programming Concepts*, Order Number SC23-4124.
- *GL3.2 Version 4 for AIX: Programming Concepts*, Order Number SC23-2612.

Ordering Publications

You can order publications from your sales representative or from your point of sale.

To order additional copies of this book, use order number SC23-4168.

Chapter 1. System Files

A file is a collection of data that can be read from or written to. A file can be a program you create, text you write, data you acquire, or a device you use. Commands, printers, terminals, and application programs are all stored in files. This allows users to access diverse elements of the system in a uniform way and gives the operating system great flexibility. No format is implied when a file is created.

Files are used for all input and output (I/O) of information in this operating system. This standardizes access to both software and hardware. Input occurs when the content of a file is modified or written to. Output occurs when the content of one file is read or transferred to another file. For example, to create a hardcopy printout of a text file, the system reads the information from the text file and writes the data to the file representing the printer.

Collections of files are stored in directories. These collections of files are often related to each other, and storing them in a structure of directories keeps them organized.

There are many ways to create, use, and manipulate files. "Files Overview" in *AIX Version 4.3 System User's Guide: Operating System and Devices* introduces the commands that control files.

Types of Files

There are three basic types of files:

regular	Stores data (text, binary, and executable).
directory	Contains information used to access other files.
special	Defines a FIFO (first-in, first-out) file or a physical device.

All file types recognized by the system fall into one of these categories. However, the operating system uses many variations of these basic types.

Regular files are the most common. When a word processing program is used to create a document, both the program and the document are contained in regular files.

Regular files contain either text or binary information. Text files are readable by the user. Binary files are readable by the computer. Binary files can be executable files that instruct the system to accomplish a job. Commands, shell scripts, and other programs are stored in executable files.

Directories contain information the system needs to access all types of files, but they do not contain the actual file data. As a result, directories occupy less space than a regular file and give the file-system structure flexibility and depth. Each directory entry represents either a file or subdirectory and contains the name of a file and the file's i-node (index node reference) number. The i-node number represents the unique i-node that describes the location of the data associated with the file. Directories are created and controlled by a separate set of commands. See "Directories" in *AIX Version 4.3 System User's Guide: Operating System and Devices* for more information.

Special files define devices for the system or temporary files created by processes. There are three basic types of special files: FIFO (first-in, first-out), block, and character. FIFO files are also called pipes. Pipes are created by one process to temporarily allow communication with another process. These files cease to exist when the first process finishes. Block and character files define devices.

Every file has a set of permissions (called access modes) that determine who can read, modify, or execute the file. To learn more about file access modes, see "File Ownership and User Groups" in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

File-Naming Conventions

The name of each file must be unique within the directory where it is stored. This insures that the file also has a unique path name in the file system. File-naming guidelines are:

- A file name can be up to 255 characters long and can contain letters, numbers, and underscores.
- The operating system is case-sensitive which means it distinguishes between uppercase and lowercase letters in file names. Therefore, `FILEA`, `FiLea`, and `filea` are three distinct file names, even if they reside in the same directory.
- File names should be as descriptive as possible.
- Directories follow the same naming conventions as files.
- Certain characters have special meaning to the operating system, and should be avoided when naming files. These characters include the following:

  ```
  / \ " ' * ; - ? [ ] ( ) ~ ! $ { } < > # @ & |
  ```

- A file name is hidden from a normal directory listing if it begins with a . (dot). When the **li** command is entered with the **-a** flag, the hidden files are listed along with regular files and directories.

The path name of a file consists of the name of every directory that precedes it in the file tree structure. Only the final component of a path name can contain the name of a regular file. All other components in a path name must be directories. Path names can be absolute or relative. See "File Path Names" in *AIX Version 4.3 System User's Guide: Operating System and Devices* to learn more about the complete name of a file within the file system.

System Files

The files in the following chapter are system files. These files are created and maintained by the operating system and are necessary for the system to perform its many functions. System files are used by many commands and subroutines to perform operations. These files can only be changed by a user with root authority.

Related Information

Files Overview in *AIX Version 4.3 System User's Guide: Operating System and Devices* introduces the basic concepts of files and directories and the commands that control them.

aliases File for Mail

Purpose

Contains alias definitions for the **sendmail** command.

Description

The **/etc/aliases** file contains the required aliases for the **sendmail** command. Do not change these defaults, as they are required by the system. The file is formatted as a series of lines in the form:

```
name: name_1, name_2, name_3,...
```

The `name:` is the name of the alias, and the `name_n` are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with a # (pound sign) are comments.

Aliasing occurs only on local names. System-wide aliases are used to redirect mail. For example, if you receive mail at three different systems, you can use the **/etc/aliases** file to redirect your mail to one of the systems. As an individual user, you can also specify aliases in your **.mailrc** file.

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 a single name.

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`.

The **/etc/aliases** file is a raw data file; the actual aliasing information is placed into a binary format in the **/etc/aliasesDB/DB.dir** and **/etc/aliasesDB/DB.pag** files by using the **newaliases** command. The **newaliases** command must be executed each time the aliases file is modified.

> **Note:** Upper case characters on the left hand side of the alias are converted to lowercase before being stored in the database manager (DBM). In the following example, mail sent to the `testalias` user alias fails, since `TEST` is converted to `test` when the second line is stored.
>
> ```
> TEST: user@machine
> testalias: TEST
> ```
>
> To preserve uppercase in user names and alias names, add the **u** flag to the local mailer description in the **/etc/sendmail.cf** file. Thus, in the example above, mail to the `testalias user alias` would succeed.

Implementation Specifics

This **/etc/aliases** file is part of Base Operating System (BOS) Runtime.

Files

/etc/aliases	Contains systemwide aliases.
/etc/aliasesDB directory	Contains the binary files created by the **newaliases** command, including the **DB.dir** and **DB.pag** files.

Related Information

The **newaliases** command, **sendmail** command.

The **.mailrc** file.

audit File for BNU

Purpose

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

Description

The **/var/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 **/var/spool/uucp/.Admin/errors** file or to standard output.

Implementation Specifics

This file is part of the Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/var/spool/uucp/.Admin/audit	Specifies the path of the **audit** file.
/var/spool/uucp/.Admin/errors	Contains a record of **uucico** daemon errors.

Related Information

The **uudemon.cleanu** command.

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

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 in the **backup** file format, and conversely, the **restore** command reads file system backups. The backup file 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 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 Version 3 by-inode 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	Indicates 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-inode 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 i-node 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 i-node as it appears on the BSD file system.
`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 a Version 3 file system.
`xix_dinode`	The real di-node from the Version 3 file system.

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

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

By-Name Format

The format of a Version 3 by-name backup is:

FS_VOLUME

FS_NAME_X (before each file)

File Data

FS_END

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

By-Inode Format

The format of a Version 3 by-inode backup follows:

TS_VOLUME

TS_BITS

TS_CLRI

TS_INODE

TS_END

A detailed description of the by-inode 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         */
#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)  /* num 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 vol header*/
#define DUMNAME      4             /* dummy name length for    */
                                   /* FS_NAME_X                */
#define FXLEN        80            /* length of file index     */
```

Implementation Specifics

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

Related Information

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

The **filesystems** file.

File Systems Overview and

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 binfile 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. Upon startup, if the **auditbin** daemon detects that the bin files require a recovery procedure, the command will prepend a **-r** to the bin file’s name in **$bin**.

> **Note:** The commands are executed by the trusted shell (TSH) when on the trusted path. 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:

   ```
   /usr/sbin/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 and append the events to the **/audit/trail.violations** file, you must include the following line in the **/etc/security/audit/bincmds** file:

```
/usr/sbin/auditselect -e "result == FAIL_AUTH || \
result == FAIL_PRIV" $bin >> /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:

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

   Adjust the printer name to fit your requirements.

Implementation Specifics

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

Files

/etc/security/audit/bincmds	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 audit events for audited objects (files).
/etc/security/audit/streamcmds	Contains auditstream commands.

Related Information

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

The **auditbin** daemon.

Setting Up Auditing in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

Security Administration,

BOOTP Relay Agent Configuration File

Purpose

Default configuration information for the BOOTP (boot protocol) relay agent program (dhcprd).

Description

The dhcprd configuration file contains entries for logging information and servers to receive BOOTP packets.

Following are the formats for the data in the configuration file.

`# Comment line`	The # character means that there is a comment from that point to the end of the line.
`numLogFiles n`	Specifies the number of log files. If 0 is specified, no log file will be maintained, and no log message is displayed anywhere. *n* is the maximum number of log files maintained as the size of the most recent log file reaches its maximum size and a new log file is created.
`logFileSize n`	Maximum size of a log file. When the size of the most recent log file reaches this value, it is renamed and a new log file is created. *n* is measured in kilobytes(KB).
`logFileName filename`	Name and path of the most recent log file. Less recent log files have the number 1 to (n - 1) appended to their names; the larger the number, the older the file.
`logItem <option name>`	One item that will be logged. Multiple of these lines are allowed. This allows for the specified logging level to be turned on. The following are option names: **SYSERR** System error, at the interface to the platform **OBJERR** Object error, in between objects in the process **PROTERR** Protocol error, between client and server **WARNING** Warning, worth attention from the user **EVENT** Event occurred to the process **ACTION** Action taken by the process **INFO** Information that might be useful **ACNTING** Who was served, and when **TRACE** Code flow, for debugging.
`server <ip address>`	The address of a server to receive the DHCP or BOOTP packet. Multiple servers may be specified, and all will receive the packet.

Example

The following example sets the logging parameters and configures two servers to receive BOOTP and DHCP packets. The servers are specified singly and with their ip addresses. The logging statements below tell the daemon to use at most four logfiles, rotate the log files after their size is 100 kilobytes of data, and place the files in the local directory and use **dhcpsd.log** as the base name. On rotation, the old file will be moved to **dhcpsd.log1**, and the daemon will start logging to an empty **dhcpsd.log**.

```
numLogFiles     4
logFileSize     100
logFileName     dhcpsd.log
logItem         SYSERR
logItem         OBJERR
logItem         PROTERR
logItem         WARNING
logItem         EVENT
logItem         ACTION
logItem         INFO
logItem         ACNTING
logItem         TRACE

server 129.35.128.43
server 9.3.145.5
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

The **dhcprd** Daemon, the **bootpd** Daemon

TCP/IP Address and Parameter Assignment - Dynamic Host Configuration Protocol (DHCP) in *AIX Version 4.3 System Management Guide: Communications and Networks*.

Problems with Dynamic Host Configuration Protocol (DHCP) in *AIX Version 4.3 System Management Guide: Communications and Networks*.

bootparams File for NFS

Purpose

Contains the list of client entries that diskless clients use for booting.

Description

The **/etc/bootparams** file for Network File System (NFS) 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
```

Files

/etc/bootparams	Specifies the path of the **bootparams** file.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

ClientHostName.info File

Purpose

Created by the Network Installation Management (NIM) software to deliver information to the boot environment of NIM client machines.

> **Note:** In AIX Version 4, this is an internal file to the Network Installation Management software and should not be modified manually.

Description

The NIM software creates the ClientHostName.info file to deliver information to the boot environment of NIM client machines. The file resides in the **/tftpboot** directory on the server of the NIM Shared Product Object Tree (SPOT), with a format of <ClientHostName>.info where <ClientHostName> is the hostname of the client machine.

After the client machine performs a network boot, it retrieves a copy of the <ClientHostName>.info file from the boot server using **tftp**. The client machine then uses the contents of the <ClientHostName>.info file to define environment variables for further processing in the boot process.

The <ClientHostName>.info file is used to support network boot for three types of NIM operations:

- Installation of the AIX Base Operating System onto standalone machines
- Initialization of diskless/dataless machines
- Diagnostics boot.

Some of the variables defined in the <ClientHostName>.info file are common to both operations while others are operation-specific.

Following are descriptions of variables that may be defined in the <ClientHostName>.info file:

> **Note:** The following are managed by the **nim** command and should not be modified by other means.

NIM_NAME	Name that identifies the client machine in the NIM environment.
NIM_HOSTNAME	Hostname that identifies the client machine.
NIM_CONFIGURATION	Machine configuration that describes the client's resource requirements. Possible values are: **standalone**, **diskless**, **dataless**.
NIM_MASTER_HOSTNAME	Hostname that identifies the NIM master in the network.
NIM_MASTER_PORT	Port number on the NIM master that should be used for NIM communications.
RC_CONFIG	File that defines the configuration procedures that the client machine should follow as it boots. Possible values are: **rc.bos_inst**, **rc.dd_boot**, **rc.diag**.
NIM_BOSINST_RECOVER	Script that initializes the bos installation environment for NIM.
SPOT	Location of the Shared Product Object Tree resource that will be used during the boot process.
ROOT	Location of the root filesystem that will be mounted by diskless/dataless machines.
DUMP	Location of the dump resource that will be mounted by diskless/dataless machines.
NIM_CUSTOM	Command to execute a NIM script during post-installation processing.
NIM_BOS_IMAGE	Image from which the Base Operating System will be installed.
NIM_BOS_FORMAT	Format of the image that will be used to install the Base Operating System.
NIM_HOSTS	ip addresses and hostnames of the NIM machines that will participate in the operation.
NIM_MOUNTS	Filesystems that will be mounted during the operation.
ROUTES	Routes from the client machine to other networks in the NIM environment. The format of each value is a colon-separated list of the network ip address, the network subnet mask, and the ip address of the gateway to the network.

Example

This example shows the contents of the file **/tftpboot/devon.austin.ibm.com.info** after a bos installation has been enabled via the following command:

```
      nim -o bos_inst -a source=rte devon
```

```
export NIM_NAME=devon
export NIM_HOSTNAME=devon.austin.ibm.com
export NIM_CONFIGURATION=standalone
export NIM_MASTER_HOSTNAME=redfish.austin.ibm.com
export NIM_MASTER_PORT=1058
export RC_CONFIG=rc.bos_inst
export
NIM_BOSINST_RECOVER="/../SPOT/usr/lpp/bos.sysmgt/nim/methods/
      c_bosinst_env -a
hostname=devon.austin.ibm.com"
export SPOT=redfish.austin.ibm.com:/spot/myspot/usr
export
NIM_CUSTOM="/../SPOT/usr/lpp/bos.sysmgt/nim/methods/c_script -a
location=redfish.austin.ibm.com:/export/nim/scripts/devon.script"
export NIM_BOS_IMAGE=/SPOT/usr/sys/inst.images/bos
export NIM_BOS_FORMAT=rte
export NIM_HOSTS=" 129.35.134.9:devon.austin.ibm.com
9.3.84.202:redfish.austin.ibm.com "
export NIM_MOUNTS="
redfish.austin.ibm.com:/lppsource/imagedir:/SPOT/usr/sys/inst.images:dir "
export ROUTES=" 9.3.84.128:255.255.255.128:129.35.128.201 "
```

Files

/tftpboot/*ClientHostName***.info**	Default location of the *ClientHostName***.info** file.

Related Information

Network Installation Management Concepts in *AIX Network Installation Management Guide and Reference*.

Command (C.*) Files for BNU

Purpose

Contains 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:

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

The *SystemName* variable indicates the name of the remote system. The *N* character represents the grade of the work. 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 characteristics are:

- 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 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 the name **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

The following are two examples of using the command (**C.***) files.

Two Send Command Files

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

   ```
   S /home/amy/f1 /var/spool/uucppublic/f2 amy -dC D.herale73655 777 lgh
   ```

   where:
   a) `S` denotes that the **uucp** command is sending the file.
   b) The full path name of the source file is `/home/amy/f1`.
   c) The full path name of the destination is `/var/spool/uucppublic/f2`, where `/var/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 `~/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 `/var/spool/uucp/hera/C.zeusN3130` send command file, produced by the **uuto** command, is as follows:

   ```
   S /home/amy/out ~/receive/msg/zeus amy -dcn D.0 777 msg
   ```

   The `S` denotes that the `/home/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.

Receive Command File

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 /home/amy/out hera!/home/amy/out2 > ~/DF"
```

produces the `/var/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 `/home/amy/out` with file `/home/amy/out2`, which is stored on the 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 /home/amy/out2 D.hera1e954fd amy - dummy 0666 amy
```

The `R` denotes a receive file. The **uucico** daemon, called by the **uux** command, gets the `/home/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp/Permissions	Describes access permissions for remote systems.
/etc/uucp/Systems	Describes accessible remote systems.
/etc/uucp/Sysfiles file	Specifies possible alternative files for **/etc/uucp/Systems**.
/var/spool/uucp/*SystemName***/D.***	Contains data to be transferred.
/var/spool/uucp/*SystemName* directory	Contains BNU command, data, and execute files.
/var/spool/uucppublic/* directory	Contains transferred files.

Related Information

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

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

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, as defined in the bin stanza, 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 file specified in the stream stanza (normally the **/etc/security/audit/streamcmds** file), is configured at the start up of the audit system.

  off
  : Stream data collection is not enabled. This is the default value.

  on
  : Stream data collection is enabled.

Note: If neither collection mode is defined or if both modes are in the **off** state, only 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
  bytethreshold = DecimalString
  eventthreshold = DecimalString
```

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. If the *binsize* parameter is set to 0, no bin switching will occur, and all bin collection will go to **bin1**.

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.

bytethreshold
: Specifies the decimal integer string that defines the approximate number of bytes written to an audit bin before a synchronous update is performed. If the **bytethreshold** is set to 0, this function is disabled. Both **bytethreshold** and **eventthreshold** can be used simultaneously. This parameter only applies to AIX Versions 4.1.4 and later.

eventthreshold
: Specifies a decimal integer string that defines the maximum number of events written to an audit bin before a synchronous update is performed. If the **eventthreshold** is set to 0, this function is disabled. Both **eventthreshold** and **bytethreshold** can be used simultaneously. This parameter only applies to AIX Versions 4.1.4 and later.

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 executed 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. Each class definition must be contained in a single line, with a new line acting as a delimiter between classes. 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
```

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 or the string `default`, 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	file name

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 = USER_Login, USER_Logout
```

These specific audit events and audit classes are described in "Setting Up Auditing" in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

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:

```
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** and **stream** stanzas of the **/etc/security/audit/config** file:

```
start:
  streammode = on
stream:
  cmds = /etc/security/audit/streamcmds
```

Implementation Specifics

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

Files

/etc/security/audit/config	Specifies the path to the file.
/etc/security/audit/objects	Contains audit events for 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.

Setting Up Auditing in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

Security Administration,

consdef File

Purpose

Enables asynchronous tty devices to be console candidates at system boot when no console device is defined or available.

Description

The **/etc/consdef** file enables tty devices such as terminals and modems to be chosen as the console device. When the console device is undefined, the system displays a message on all natively attached graphics displays and the tty on native serial port S1. The console device is undefined when:

- The system is first installed and started.
- The console definition has been deleted from the ODM database.
- The console device has been physically removed from the system.

If any of these conditions occur, the system displays the following message:

```
******* Please define the System Console. *******
Type a Number and press <Enter> to use this terminal as the system console.
```

For high function terminals (HFTs)graphics displays, the *Number* variable refers to a function key. For asynchronous ttys, this variable is a number.

The selected item becomes the system console. To choose a non-default tty device as the system console, you must first configure the **/etc/consdef** file. This file contains stanzas that define various console attributes. Each line, or entry, in a stanza must take the form of *Attribute*=*Value*, and the line must not exceed 80 characters. The following attributes must be defined for each terminal device:

connection Identifies the type of tty interface. Valid values are `rs232` and `rs422`.

location Specifies the location code of the terminal. Location codes of `00-00-S1-00` or `00-00-S2-00` indicate that the tty device is attached to the S1 or S2 serial port, respectively. Any other location code indicates the tty device is attached to an adapter card other than the standard I/O planar. You can display valid location values with the **lsdev -C | grep tty** command.

You can also specify other terminal attributes such as **speed**, **bpc**, **stops**, **parity**, and `term`. If you do not define these attributes, the system uses the default values stored in the ODM database. The **consdef** file contains a sample stanza for the S1 port. To enable this stanza, or parts of it, remove the comment delimiters (#) from each applicable line.

Examples

To display the console selection message on the ttys attached to the S1 and S2 ports:

```
ALTTTY:
   connection=rs232
   location=00-00-S1-00
   speed=9600
   bpc=8
   stops=1
   parity=none
   term=ibm3163

ALTTTY:
   connection=rs232
   location=00-00-S2-00
   speed=9600
   bpc=8
   stops=1
   parity=none
   term=ibm3163
```

Note: For backward compatibility, the `ALTTTY:` keyword is not required for the first entry.

Implementation Specifics

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

Files

/etc/consdef	Specifies the path of the **consdef** file.
/dev/console	Provides access to the system console.

Related Information

The **chcons** command.

The **lsdev** command.

The **console** special file.

Data (D.*) Files for BNU

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:

/var/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 **/var/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 **/var/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp/Systems	Describes accessible remote systems.
/var/spool/uucp/*SystemName* directory	Contains BNU command, data, and execute files.
/var/spool/uucp/*SystemName***/C.***	Contains instructions for file transfers.
/var/spool/uucp/*SystemName***/TM.***	Stores data files temporarily after they have been transferred to a remote system.
/var/spool/uucppublic/* directory	Contains files that the BNU program has transferred.

Related Information

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

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

/dev/hty File

Purpose

Defines the Network Terminal Accelerator adapter tty interface.

Description

The **/dev/hty*** device files define, for the host computer, the interface-to-host adapter communication channels. For each I/O device connected to the host computer through a host adapter, there must be a **/dev/hty*** device file created to allow communication between the host computer and the I/O device.

To allow for future expansion, there may be more **/dev/hty*** files than actual physical devices connected through the host adapter.

The hty ports are functionally equivalent to **/dev/tty*** device files. The minor number corresponds to the channel number, as defined in the **hty_config** file.

Implementation Specifics

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

Files

/dev/hty	Specifies the path to the file.
/dev/rhp*	Adapter raw device.

Related Information

/dev/rhp File

Purpose

Defines the Network Terminal Accelerator adapter raw interface.

Description

The **/dev/rhp*** device files define, for the host computer, the interface to the host adapters. For each host adapter installed in the host computer, there must be a **/dev/rhp*** device file created in order to allow communication between the host computer and the host adapter board.

The **/dev/rhp*** device file corresponding to a respective host adapter is used as an argument in many of the utility programs.

Files

/dev/rhp	Specifies the path to the file
/dev/hty	Defines the Network Terminal Accelerator adapter tty interface.

Implementation Specifics

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

Related Information

DHCP Client Configuration File

Purpose

Default configuration information for the Dynamic Host Configuration Protocol (DHCP) client program (dhcpcd).

Description

The dhcpcd configuration file contains entries for logging information, requested options, interfaces to configure, and other items.

Following are the formats for the data in the configuration file.

`# Comment line`	The # character means that there is a comment from that point to the end of the line.
`numLogFiles n`	Specifies the number of log files. If 0 is specified, no log file will be maintained and no log message is displayed anywhere. *n* is the maximum number of log files maintained as the size of the most recent log file reaches its maximum size and a new log file is created.
`logFileSize n`	Maximum size of a log file. When the size of the most recent log file reaches this value, it is renamed and a new log file is created. *n* is measured in kilobytes(KB).
`logFileName filename`	Name and path of the most recent log file. Less recent log files have the number 1 to (n - 1) appended to their names; the larger the number, the older the file.

logItem <option name>

One item that will be logged. Multiple of these lines are allowed. This allows for the specified logging level to be turned on. The following are option names:

SYSERR
System error, at the interface to the platform

OBJERR
Object error, in between objects in the process

PROTERR
Protocol error, between client and server

WARNING
Warning, worth attention from the user

EVENT
Event occurred to the process

ACTION
Action taken by the process

INFO
Information that might be useful

ACNTING
Who was served, and when

TRACE
Code flow, for debugging.

interface <ifName>

The interface to configure DHCP on. This may be the interface that is to be configured. Multiples of these are allowed. There is a special entry, any. This tells the DHCP client to configure the first one it finds and completes successfully. If the any option is used, there should not be any other interface specified. The interface statement may be immediately followed by a pair of curly braces, in which the options requested for this interface can be specified. Options requested within interface curly braces apply only to this interface. See DHCP Server Configuration File for a list of options and formats.

clientid <MAC | HOSTNAME>

Specifies the client id to use in all communication with the server. MAC denotes that the hardware address for the particular interface should be used as the client id. HOSTNAME denotes that the domain host name should be used as the client id. The default is MAC.

sniffer <exec string>

Specifies a string enclosed in quotes, indicating a program to execute to detect hardware failure/recovery for an interface. The dhcp client will look for signal 23(SIGIO) to indicate that the network interface is up and signal 16(SIGURG) to indicate that the network interface is down.

option <code> [<value>] [exec <string>]

	Specifies an option requested by this client. Its scope is determined by whether it is inside a set of curly braces for a particular interface, or if it is outside all curly braces. If outside, it applies to all interfaces. `code` is the option code of the option requested. `value` is the requested value for that option. This value is passed to the server with the option. The value is not required. The keyword `exec` denotes a string following which should be executed if this option is returned by the server. This string is expected to be an executable shell script or program. An "%s" may be included in the string. If present, the value returned by the server will be provided in ascii.
`vendor`	Specifies the special syntax for the specification of the vendor extensions field. It is followed by a set of curly braces. Inside the curly braces, the options and values for the vendor extensions field are specified. The exec string on an option inside the vendor extensions options is not valid. It is ignored.
`reject <code>`	Specifies that if this option code is returned by the server, this option should be ignored by the client. Its value should not be used.
`otherOptions <accept | reject>`	Specifies how all other options should be handled by the client. This refers to any options not specifically requested with an "option" statement or rejected with a "reject" statement. The default is that all options are accepted.

`updateDNS <string>`

A string enclosed in quotes, indicating a program to execute to update the DNS server with the new inverse mapping for the IP address and names served by **dhcp**. This string should include four %s's to indicate the placement of the following information from the **dhcp** client:

`hostname`
: Value of option 12. The value returned by the **dhcp** server is used, if one is supplied. Else, if the client specified a value in *this* file, the client-requested value is used. If neither the client specified a requested hostname nor the server supplied one, this exec string will not be executed.

`domainname`
: Value of option 15. The value returned by the **dhcp** server is used, if one is supplied. Else, if the client specified a value in *this* file, the client-requested value is used. If neither the client specified a requested hostname nor the server supplied one, a null string (" ") will be supplied by **dhcp**. Therefore, this value is optional.

`Ip Address`
: IP address leased to this client by the server. The string is supplied in dotted notation, for example, 9.2.23.43.

`leasetime`
: Lease time granted by the server. This string is a decimal number representing the number of seconds of the lease.

These values are output by **dhcp** in this order:

```
hostname domainname Ip Address leasetime
```

A script **/usr/sbin/dhcpaction** has been provided with this function, as well as actions to help NIM interact with DHCP clients. Run the script as follows:

```
/usr/sbin/dhcpaction hostname domainname ipaddress
leasetime < A | PTR | BOTH | NONE > NONIM
```

The first four parameters are what will be used to update the DNS server. The fifth parameter tells **dhcpaction** to update the A record, the PTR record, or both, or none. The options are `A`, `PTR`, `BOTH`, `NONE`. The sixth parameter is used to tell servers that NIM is being used, and processing needs to be done when a client changes address. The options for this are NIM and NONIM. On clients, this must be set to NONIM.

An example follows:

```
updateDNS "/usr/sbin/dhcpaction %s %s %s %s PTR
NONIM 2>&1 >>/tmp/updns.out"
```

Example

This example tells the **dhcpcd** daemon to use log files of a maximum of 100Kb in size and at most four of them.

The base name for the log files is **/usr/tmp/dhcpsd.log**. The user also would like to only log four of the nine possible log entry types. The user also specified a string to use for updating the Dynamic Domain Name Server. The user also specified that the `clientid` to the server should be based on the mac-address of the interface adapter that is trying to be configured. The user also specified that all options should be accepted and instantiated (`otheroptions accept`), except for option 9 (`reject 9`).

The options the user specified were the domain (option 15), but since this option is global to the interface keywords, it applies to both interfaces.

Inside each interface, the hostname is specified with option 12.

```
numLogFiles     4
logFileSize     100
logFileName     /usr/tmp/dhcpsd.log
logItem         SYSERR
logItem         OBJERR
logItem         PROTERR
logItem         TRACE

updateDNS "nsupdate -h%s -d%s -i% %s"

clientid MAC
otheroptions accept
reject 9

option 15 "austin.ibm.com"

interface en0
{
        option 12 "e-chisos"
}

interface tr0
{
        option 12 "t-chisos"
}
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

The **dhcpcd** Daemon

The **DHCP Server Configuration** File

TCP/IP Address and Parameter Assignment - Dynamic Host Configuration Protocol (DHCP) in *AIX Version 4.3 System Management Guide: Communications and Networks*.

Problems with Dynamic Host Configuration Protocol (DHCP) in *AIX Version 4.3 System Management Guide: Communications and Networks*.

DHCP Server Configuration File

Purpose

Defines default configuration information for the Dynamic Host Configuration Protocol (DHCP) server program (dhcpsd).

Description

The dhcpsd configuration file contains entries for logging information, options to return, machines to configure, and other items.

Following are the formats for the data in the configuration file.

`# Comment line`
: The # character means that there is a comment from that point to the end of the line.

`## "Name of Resource" "<Keyword> <value> <value> ..."`
: The ## characters denote a named resource. This is used by the **dhcpsconf** program to allow the user to create specific resources. The data is stored in the server file so that it can be read in with the configuration file and displayed as the name and not the value in the viewing window of **dhcpsconf**.

  The format of the ## line is a quoted string that is the name of the resource followed by a double-quoted string representing a valid possible line for a configuration file. The second quoted string should be syntactically correct for a line in a DHCP server configuration file. The keyword can only be `option`, `network`, `subnet`, `class`, and `client`.

`### "DHCP Server" "Any line from a server file"`
: The ### characters denote a server configuration file. This allows for multiple server files to be saved in one file. The **dhcpsconf** program uses this to present multiple server datasets in a master. This would be useful, if you were to define a network with 10 servers and wanted to save all the server information in one file and maintain a default server. The default server would go into the master file, and the servers would be saved in the master file with the ### characters. The **dhcpsconf** program has a function that allows you to create a specific server configuration out of the master file.

`numLogFiles n`
: Specifies the number of log files. If 0 is specified, no log file will be maintained and no log message is displayed anywhere. *n* is the maximum number of log files maintained as the size of the most recent log file reaches its maximum size and a new log file is created.

`logFileSize n`
: Maximum size of a log file. When the size of the most recent log file reaches this value, it is renamed and a new log file is created. *n* is measured in kilobytes(KB).

`logFileName filename`
: Name and path of the most recent log file. Less recent log files have the number 1 to (n - 1) appended to their names; the larger the number, the older the file.

`logItem <option name>`

One item that will be logged. Multiple of these lines are allowed. This allows for the specified logging level to be turned on. The following are option names:

SYSERR	System error, at the interface to the platform
OBJERR	Object error, in between objects in the process
PROTERR	Protocol error, between client and server
WARNING	Warning, worth attention from the user
EVENT	Event occurred to the process
ACTION	Action taken by the process
INFO	Information that might be useful
ACNTING	Who was served, and when
TRACE	Code flow, for debugging.

`clientrecorddb <filename>`

This is the path to a file to substitute for **/etc/dhcps.cr**. Configurations that support a large number of addresses should set **clientrecorddb** and **addressrecorddb** database files in a file system with substantial free space.

`addressrecorddb <filename>`

This is the path to a file to substitute for **/etc/dhcps.ar**.

`network <Network address> [<Subnet Mask>|<range>]`

Specifies one network administered by this server. Network address is the address of this network. This address is specified in the dotted notation (for example, 9.0.0.0, 128.81.0.0, or 192.81.20.0). Full four-byte value should be specified (for example, 9, 128.81, or 192.81.20 is not legal).

Network address may optionally be followed by the subnet mask, a range, or nothing.

If a subnet mask is specified, one or more subnet statements should appear in the succeeding lines within a pair of curly braces. The subnet mask may be specified either in the dotted notation (for example, 255.255.255.128) or as a number indicating the number of 1 bits in the mask (for example, 25, which is equivalent to 255.255.255.128). The means that a network is not a collection of all subnet for a network, but all subnets with the same length subnet for that network "prefix."

If a range is specified, it determines, within the network, the range of hosts that are administered by this server, and it implies that there is no subnetting. A range is specified by the host addresses, in the dotted notation, at the lower end and the higher end of the range, respectively, separated by a hyphen with no spaces before or after it (for example, 192.81.20.1-129.81.20.128). A range must encompass all addresses to be administered because multiple network statements to define the same network are not allowed. Use the "client" statement to exclude any addresses in the range that the server should not administer.

If nothing is specified after Network address, all hosts in that network are administered by this server.

A network statement may be immediately followed by a pair of curly braces, in which parameters (for example, options) particular to this network can be specified.

`subnet <Subnet address> [<range>]`

One or more subnet statements are enclosed by a pair of curly braces that immediately follows a network statement with subnet mask. A subnet statement specifies one subnet within that network.

Subnet address is the address of this subnet. This address is specified in the dotted notation (for example, 9.17.32.0 or 128.81.22.0).

Subnet address may be followed by a range or nothing.

If a range is specified, it determines, within the subnet, the range of hosts that are administered by this server. A range is specified by the host addresses, in the dotted notation, at the lower end and the higher end of the range, respectively, separated by a hyphen with no spaces before or after it. A range must encompass all addresses to be administered since multiple subnet statements to define the same subnet are not allowed. Use the "client" statement to exclude any addresses in the range which the server should not administer.

If nothing is specified after Subnet address, all hosts in that subnet are administered by this server.

The ranges in two servers administering the same subnet cannot overlap. Otherwise, two hosts may be assigned the same address.

A subnet statement may be immediately followed by a pair of curly braces, in which parameters (for example, options) particular to this subnet can be specified.

`class <class_name> [<range>]`

Specifies a class. The class name is a simple ascii string. A class's scope is determined by the curly braces in which it is enclosed. If it is outside all curly braces, then its scope is the entire file.

A class name may be followed by a range or nothing. If a range of Ip Addresses is specified, then only addresses in that range will be assigned to clients who request this class. Note that clients who request this class, for which the subnet does not match the range, will not be processed. Bad addresses will not be given out by the server. If an address range is not specified, then addresses will be given to clients using the usual rules of assignment (by network clauses).

The class statement may be immediately followed by a pair of curly braces, in which the options particular to this class can be specified. A class may be defined within the curly braces of a subnet, but a subnet may not be defined within the curly braces of a class.

Options set up in the network or subnet containing a class definition will also apply to the class.

`client <id_type> <id_value> <address>`

Specifies a definition of client/address processing.

<id_type> is 0 for a string, otherwise it is one of the hardware types defined in RFC 1340 (for example, 6 for IEEE 802 networks.)

<id_value> is a character string for <id_type>=0. Typically, this would be a domain name. For a non-zero <id_type>, the <id_value> is a hexadecimal string representing the hardware address of the client.

> **Note**: An <id_type> of 0 and an <id_value> of 0 indicates that the <address> specified should not be distributed by this server.

The <address> can be the string "none" to indicate that the client with <id_type> and <id_value> should not be serviced by this server. The <address> can be the string "any" to indicate that the server should choose an appropriate address for this client. The <address> can be an internet address in dotted notation (for example, 9.2.15.82). This will be the Ip address given to the particular client specified by <id_type> and <id_value>. As mentioned above, an <id_type> of 0 and an <id_value> of 0 indicates that the <address> specified should not be distributed by this server.

> **Note**: If a client is configured in this way on the server, then any class information requested by the client will be ignored. No class-specific information will be processed for these clients.

The client statement may be immediately followed by a pair of curly braces, in which the options particular to this client can be specified.

A client statement with an address specified that is not part of the address pool specified in a network/subnet elsewhere in this file must contain the subnet mask option(1). For all other clients, the server will compute the subnet mask option to send the client based on the network/subnet definitions.

> **Note**: All clients inherit all globally defined options. A client defined in a network scope will inherit options defined for that network. A client defined in a subnet scope, will inherit options defined for that subnet and encompassing network.

A class definition inside a client scope is not allowed.

The client statement may be used to configure **bootp** clients. To do this, specify all the **bootp** options using the option syntax defined below. In addition, specify an infinite lease time in the client scope with "option 51 0xffffffff". DHCP options will not be served to the **bootp** client.

option <code> <value>

This parameter specifies the value of an option defined in "DHCP Options and BOOTP Vendor Extensions" (RFC 1533) and supported by this server.

An option is specified by the "option" keyword followed by the option code of this option and its data field, in a single line. One or more of this parameter may be specified.

The scope within which an option applies is delimited by a pair of curly braces ({, }) surrounding this parameter.

Two or more options with the same option code may be specified. Their data fields are concatenated in a single option in a packet generated by the server if the options have the same scope or one's scope includes that of another.

Some of the defined options do not need to be specified by this parameter. These options are either mandated by the protocol or this implementation to be present in proper packets, or only generated by a client. These options are:

Option Code	Name
0	Pad Option
255	End Option
1	Subnet Mask
50	Request IP Address
51	IP Address Lease Time
52	Option Overload
53	DHCP Message Type
54	Server Identifier
55	Parameter Request List
57	Maximum DHCP Message Size
58	Renewal (T1) Time Value
59	Rebinding (T2) Time Value
60	Class identifier of client
61	Client identifier.

The other options may be specified by this parameter.

When specifying an option, its data field takes one of the following formats:

IP Address	xxx.xxx.xxx.xxx
IP Addresses	[xxx.xxx.xxx.xxx ...]
IP Address Pair	[ip address:ip address]
IP Address Pairs	[[ip address:ip address] ...]
Boolean	[0, 1]
Byte	[-128, 127]
Unsigned Byte	[0, 255]
Unsigned Bytes	[[0, 255] [0, 255] ...]
Short	[-32768, 32767]
Unsigned Short	[0, 65535]
Unsigned Shorts	[[0, 65535] [0, 65536]
Long	[-2147483648, 2147483647]
Unsigned Long	[0, 4294967295]
String	"Value Here"

Note: All IP addresses are specified in dotted-decimal form.

Each of the defined options is listed below by its code and name, followed by the format of its data field. These are specified in latest Vendor Extensions RFC.

Code	Name	Data Field Format and Notes
0	Pad Option	No need to specify
255	End Option	No need to specify
1	Subnet Mask	Unsigned Long
2	Time Offset	Long
3	Router Option	IP Addresses
4	Timer Server Option	IP Addresses
5	Name Server Option	IP Addresses
6	Domain Name Server Option	IP Addresses
7	Log Server Option	IP Addresses
8	Cookie Server Option	IP Addresses
9	LPR Server Option	IP Addresses
10	Impress Server Option	IP Addresses
11	Resource Location Server Option	IP Addresses
12	Host Name Option	String
13	Boot File Size Option	Unsigned Short
14	Merit Dump File	String
15	Domain Name	String
16	Swap Server	IP Address
17	Root Path	String
18	Extensions Path	String

IP Layer Parameters per Host

Code	Name	Data Field Format and Notes
19	IP Forwarding Enable/Disable Option	Boolean
20	Non-local Source Routing Enable/Disable Option	Boolean
21	Policy Filter Option	IP Address Pairs
22	Maximum Datagram Reassembly Size	Unsigned Short
23	Default IP Time-to-live	Unsigned Byte
24	Path MTU Aging Timeout Option	Unsigned Long
25	Path MTU Plateau Table	Unsigned Shorts

IP Layer Parameters per Interface

Code	Name	Data Field Format and Notes
26	Interface MTU Option	Unsigned Short
27	All Subnets are Local Option	Boolean
28	Broadcast Address Option	IP Address
29	Perform Mask Discovery Option	Boolean
30	Mask Supplier Option	Boolean
31	Perform Router Discovery Option	Boolean
32	Router Solicitation Address Option	IP Address
33	Static Route Option	IP Address Pairs

Link Layer Parameters per Interface

Code	Name	Data Field Format and Notes
34	Trailer Encapsulation Option	Boolean
35	ARP Cache Timeout Option	Unsigned Long
36	Ethernet Encapsulation Option	Boolean

TCP Parameters

Code	Name	Data Field Format and Notes
37	TCP Default TTL Option	Unsigned Byte
38	TCP Keepalive Interval Option	Unsigned Long
39	TCP Keepalive Garbage Option	Boolean

Application and Service Parameters

Code	Name	Data Field Format and Notes
40	NIS Domain Option	String
41	NIS Option	IP Addresses
42	Network Time Protocol Servers Option	IP Addresses
43	Vendor Specific Information	Unsigned Bytes
44	NetBIOS over TCP/IP Name Server Option	IP Addresses
45	NetBIOS over TCP/IP Datagram Distribution Server	IP Addresses
46	NetBIOS over TCP/IP Node Type Option	Unsigned Byte
47	NetBIOS over TCP/IP Scope Option	Unsigned Bytes
48	X Window System Font Server Option	IP Addresses
49	X Window System Display Manager Option	IP Addresses

DHCP Extensions

Code	Name	Data Field Format and Notes
50	Request IP Address	No need to specify
51	IP Address Lease Time	Unsigned Long
52	Option Overload	No need to specify
53	DHCP Message Type	No need to specify
54	Server Identifier	No need to specify
55	Parameter Request List	No need to specify
56	Message	String
57	Maximum DHCP Message Size	No need to specify
58	Renewal (T1) Time Value	No need to specify
59	Rebinding (T2) Time Value	No need to specify
60	Class Identifier of Client	Generated by client
61	Client Identifier	Generated by client

BOOTP Specific Options

Code	Name	Data Field Format and Notes
sa	Server Address for the BOOTP client to use	IP Address
bf	Bootfile for the BOOTP client to use	String
hd	Home Directory for the BOOTP client to search for the bootfile	String

Following is an example of BOOTP specific options:

```
option sa 1.1.2.2
option hd "/vikings/native"
option bf "bootfile.asdg"
```

Other option numbers may be specified, up to a maximum of 255. The options not listed above must be specified with the unsigned byte list type. Following is an example:

```
option 178 01 34 53 # Means place tag 178 with value
0x013553
```

```
leaseTimeDefault <amount>[<unit>]
```

Specifies the default lease duration for the leases issued by this server. In the absence of any more specific lease duration (for example, lease duration for specific client(s) or class of clients), the lease duration specified by this parameter takes effect.

The amount is specified by a decimal number. The unit is one of the following (plural is accepted):

- year
- month
- week
- day
- hour
- minute (default if unit is absent)
- second

There is at least one white space in between the amount and unit. Only the first amount following the keyword has effect.

If this parameter is not specified, the default lease duration is one (1) hour.

This parameter should appear outside of any pair of curly braces, for example, it applies to all leases issued by this server.

> **Note:** This keyword only applies to the default for all addresses. To specify a specific lease time for a subnet, network, class or client, use the usual "option 51 value" to specify that lease time (in seconds).

```
leaseExpireInterval <amount> [<unit>]
```

Specifies the time interval at which the lease expiration condition is examined, and if a running lease meets such condition, it is expired. The value of this parameter applies to all leases administered by this server.

The amount is specified by a decimal number. The unit is one of the following (plural is accepted):

- year
- month
- week
- day
- hour
- minute (default if unit is absent)
- second

There is at least one white space in between the amount and unit. Only the first amount following the keyword has effect.

If this parameter is not specified, the default interval is one (1) minute.

This parameter should appear outside of any pair of curly braces, for example it applies to all leases issued by this server.

The value of this parameter *should* be in proportion with that of parameter `leaseTimeDefault` so that the expirations of leases are recognized in time.

`supportBOOTP [yes | no]`

Indicates to the server whether or not to support requests from BOOTP clients.

If `yes` is specified, the server will support BOOTP clients.

If the value field is not a `yes`, or the keyword is omitted, the server will not support BOOTP clients.

The scope of this parameter covers all the networks and subnets administered by this server.

If the server previously supported BOOTP clients and has been reconfigured not to support BOOTP clients, the address binding for a BOOTP client established before the reconfiguration, if any, will still be maintained until the time when that BOOTP client sends a request again (when it is rebooting.) At that time, the server will not respond, and the binding will be removed.

`supportunlistedClients [yes | no]`

Indicates to the server whether or not to support requests from clients that are not specifically configured with their own individual client statements in the server.

If `yes` is specified, the server will support unlisted clients.

If the value field is anything other than `yes`, the server will not support unlisted clients.

If this keyword is not found in the file, the server *will* support clients not specifically configured with a client statement.

`updateDNS <string>`

A string enclosed in quotes, indicating a program to execute to update the DNS server with the new inverse mapping for the IP address and names served by **dhcp**. This string should include four %s's to indicate the placement of the following information from the **dhcp** client:

`hostname`
: Value of option 12. The value returned by the **dhcp** server is used, if one is supplied. Else, if the client specified a value in *this* file, the client-requested value is used. If neither the client specified a requested hostname nor the server supplied one, this exec string will not be executed.

`domainname`
: Value of option 15. The value returned by the **dhcp** server is used, if one is supplied. Else, if the client specified a value in *this* file, the client-requested value is used. If neither the client specified a requested hostname nor the server supplied one, a null string (" ") is supplied by **dhcp**. This may cause the update of address records to fail.

`Ip Address`
: IP address leased to this client by the server. The string is supplied in dotted notation, for example, 9.2.23.43.

`leasetime`
: Lease time granted by the server. This string is a decimal number representing the number of seconds of the lease.

These values are output by **dhcp** in this order:

```
hostname domainname Ip Address leasetime
```

A script **/usr/sbin/dhcpaction** has been provided with this function, as well as actions to help NIM interact with DHCP clients. Run the script as follows:

```
/usr/sbin/dhcpaction hostname domainname ipaddress
leasetime < A | PTR | BOTH | NONE > < NONIM | NIM >
```

The first four parameters are what will be used to update the DNS server. The fifth parameter tells **dhcpaction** to update the A record, the PTR record, or both, or none. The options are A, PTR, BOTH, NONE. The sixth parameter is used to tell servers that NIM is being used, and processing needs to be done when a client changes address. The options for this are NIM and NONIM.

An example follows:

```
updateDNS "/usr/sbin/dhcpaction %s %s %s %s PTR
NONIM 2>&1 >>/tmp/updns.out"
```

Examples

1. In this example, we are setting up a server with a default lease time of 30 minutes. This means that any address that doesn't explicitly have a lease time set in a network, class, client, or subnet scope, will get 30 minutes. We are also setting the time between server address expiration checks to 3 minutes. This means that every 3 minutes, the server will check to see if an address has expired and mark it as expired. We are also saying the server should accept BOOTP requests and accept any client that matches the normal address assignment scheme. The normal address

assignment scheme means that an address and options are assigned based on the network/subnet that the client is on.

We are also setting up two global options that should apply to all clients we serve. We are saying that there is a printer at 10.11.12.13 for everyone to use and the global domain name is `dreampark`. We are defining one network that has subnetting on the first 24 bits.

Thus, the network we are defining has some number of subnets and all the subnets we are specifying in this network scope have netmask of 255.255.255.0. Under that network, we are defining some options for that network and some subnets. The subnets define the actual addresses available for distribution. There are two subnets. Inside the second subnet, there is a class. The class information only applies to hosts on the second subnet that request that class. If that class is asked for the host, it will get two netbios options. If the address is in the first subnet, it will get the options in the subnet clause, which are nothing. If the host is in the second subnet, it will get all the options in the clause for the second subnet. If it also has the class, it will get the class options. If options are repeated with the same scope or a sub-scope, these options are concatenated together and set as one option. All hosts given an address from one of the two subnets will receive the options that are in the network scope.

```
leaseTimeDefault                   30 minutes
leaseExpireInterval                3 minutes
supportBOOTP                       yes
supportUnlistedClients             yes

option 9         10.11.12.13                  # printer for all
option 15        dreampark                    # domain
name

network 9.0.0.0 24
{
          subnet 9.2.218.0     9.2.218.1-9.2.218.128
          subnet 9.67.112.0    9.67.112.1-9.67.112.64
          {
            option 28          9.67.112.127             # broadcast address
            option 9           9.67.112.1               # printer 1
            option 9           9.67.112.2               # printer 2
            option 15          sandbox.                 # domain name
            class netbios_host
            {
                               #Netbi ov tcp/ip name server
                               option 44 9.67.112.125
                               Netbi over tcp/ip node type
                               option 46 2
                   }
            }

            option 15          toyland                   # domain name
            option 9           9.68.111.128              # printer 3
            option 33          1.2.3.4:9.8.7.1           # route to the moon
            option 33          5.6.7.8:9.8.7.2           # route to the mars
            # routes to black holes
            option 3           11.22.33.44    55.66.77.88
}
```

2. In this example, we see the output of the **dhcpsconf** command. This format is more used by the **dhcpsconf** GUI to store information. This format allows for multiple configurations. The **dhcpsconf** GUI can in turn generate the specific server files for an individual server. The file

specifies two of DHCP Servers, `Greg` and `Fred`. Each contain the definitions for the two servers. The **dhcpsconf** command can generate files specifically for `Greg` or `Fred`. The **dhcpsconf** command will also use the named resources (## sections) to display network pieces that have been named by the administrator.

The DHCP server `Greg` is responsible for network 9.3.145.0, subnet mask 255.255.255.192. The DHCP server `Fred` is responsible for network 9.3.146.128, subnet mask 255.255.255.240. Each server provides its own domain name. Other options named and unnamed may be placed in the server's configuration section.

Note: This format is used by **dhcpsconf**, which generateS the appropriate configuration files for DHCP servers `Greg` and `Fred`.

```
# Named resources Section
## "Network 1 Subnet Netmask" "option 1 255.255.255.192"
## "Network 2 Subnet Netmask" "option 1 255.255.255.240"
## "Network 1 Domain Name" "option 15 "bizarro.austin.ibm.com""
## "Network 2 Domain Name" "option 15 "superman.austin.ibm.com""
## "Network 1 Network" "network 9.3.145.0 26"
## "Network 2 Network" "network 9.3.146.128 27"

### "DHCP Server Greg" "logItem SYSERR"
### "DHCP Server Greg" "numlogfiles 6"
### "DHCP Server Greg" "logfilesize 100"
### "DHCP Server Greg" "logfilename /usr/tmp/dhcpgreg.log"
### "DHCP Server Greg" "network 9.3.145.0 26"
### "DHCP Server Greg" "{"
### "DHCP Server Greg" "option 15 "bizarro.austin.ibm.com""
### "DHCP Server Greg" "}"
### "DHCP Server Fred" "logItem SYSERR"
### "DHCP Server Fred" "logItem OBJERR"
### "DHCP Server Fred" "numlogfiles 3"
### "DHCP Server Fred" "logfilesize 50"
### "DHCP Server Fred" "logfilename /usr/tmp/dhcpfred.log"
### "DHCP Server Fred" "network 9.3.146.128 27"
### "DHCP Server Fred" "{"
### "DHCP Server Fred" "option 15 "superman.austin.ibm.com""
### "DHCP Server Fred" "}"
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

The **dhcpsd** daemon, the **dhcpsconf** command

The **DHCP Client Configuration** File

TCP/IP Address and Parameter Assignment - Dynamic Host Configuration Protocol (DHCP) in *AIX Version 4.3 System Management Guide: Communications and Networks*.

Problems with Dynamic Host

dir File

Purpose

Describes the format of a directory.

Syntax

#include <sys/dir.h>

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 the control of the operating system.

The 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 follows.

> **Note:** This structure is a 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.

The **DIRSIZ (***dp***)** macro gives the amount of space required to represent a directory entry. The *dp* argument is a pointer to a **direct** structure.

Implementation Specifics

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

Related Information

The **dirent.h** file, **filsys.h** file, **inode** file.

The **opendir**, **readdir**, **telldir**, **seekdir**, **rewindir**, or **closedir** subroutine.

File Systems Overview in *AIX Version 4.3 System Management Concepts: Operating System and Devices*.

Directory Overview in *AIX Version 4.3 System User's Guide: Operating System and Devices*.

Files Overview in *AIX Version 4.3 System User's Guide: Operating System and Devices*.

dsinfo File

Purpose

Contains the terminal descriptions for the **Dynamic Screen** utility.

Description

The **dsinfo** file is a database of terminal descriptions used by the **Dynamic Screen** utility. A terminal description typically contains the following configuration information:

- Keys defined for specific use with the **Dynamic Screen** utility and their function
- Number of pages of screen memory available to the terminal
- Code sequences that must be sent or received to access and use Dynamic Screen features

The **dscreen** command reads the appropriate configuration information from the **dsinfo** file to start the **Dynamic Screen** utility.

Entry Format

Line entries in the **dsinfo** file consist of a number of definition fields separated by commas. The first-line field entries are alternate screen names for the terminal. The screen name fields are separated by a | (pipe symbol).

Other line fields are strings describing the capabilities of the terminal definition to the **Dynamic Screen** utility. The following escape codes are recognized within these strings:

Escape Code	Meaning
\E,\e	Escape
\n,\l	New line
\r	Carriage return
\t	Tab
\b	Backspace
\f	Form feed
\s	Space
nnn	Character with octal value *nnn*
^*x*	Ctrl-*x* for any appropriate *x*.

Any other character preceded by a \ (backslash) yields the character itself.

Strings must be entered as the *type=string* parameter, where *type* is the string type and *string* is the string value.

If information is not entered into a string field, a comma is still used to designate the existence of the field.

String Types and String Values

The following string types are available:

dsk*x* Describes the action assigned to a key. This string type contains 4 characters. The 4th character indicates the action to be taken when the keystroke is received by the screen:

Key Type	Action
dskb	Block input and output.
dskc	Start a new screen.
dske	End the **Dynamic Screen** utility (exit code 0).
dskl	List keys and actions.
dskp	Switch to previous screen.
dskq	Quit **Dynamic Screen** utility (exit code 1).
dsks	Select a specific screen.

Currently, the only valid **dsk** string type endings are b, c, e, l, p, q, and s. Any other key definitions used at this time are interpreted as null values and cause no internal Dynamic Screen action for the terminal definition. Other keys may be assigned values within the **Dynamic Screen** utility at a later time.

> **Note:** The **dskn** string type (n for null or no operation) is guaranteed not to be used for any function assignments in future versions. It is recommended that the **dskn** string type be used instead of other null characters when no internal Dynamic Screen action is desired for a terminal definition.

The value string for each **dsk***x* string type has three substrings, separated by a | (pipe symbol). (To include a | in one of the substrings, use \| [backslash, pipe symbol].)

The first substring is the sequence of characters the terminal sends when the key is pressed. The second substring is a label for the key as displayed in the key listing (for example, the Shift-F1 key sequence). The third substring is a sequence of characters the **Dynamic Screen** utility sends to the terminal when the key is pressed, before performing the requested action.

dsp Describes a physical screen in the terminal. A **dsp** string type must be present for each physical screen in the terminal.

The value string for each physical screen has two substrings, separated by a `|` (pipe symbol). (To include a `|` in one of the substrings, use `\|` [backslash, pipe symbol].)

The first substring is the sequence of characters to send to the terminal to display and output to the particular named physical page on the terminal. The second substring is usually set to clear the screen sequence. It is sent under the following two conditions:

- The creation of new terminal session
- More terminals are running than there are physical screens.

  If your selection of a terminal causes the **Dynamic Screen** utility to reuse one of the physical screens, the clear-the-screen sequence is sent to the screen to indicate that the screen content does not match the output of the terminal connected to it.

  > **Note:** Running with more terminals than there are physical screens is not recommended. Avoid this situation by defining no more screen selection keys (`dsks=`...) than physical screens (`dsp=`...).

dst Adjusts the **Dynamic Screen** utility's input timeout. The value of the string must be a decimal number. The timeout value is in tenths of a second and has a maximum value of 255. The default timeout value is 1, or one tenth of a second.

When the **Dynamic Screen** utility recognizes a prefix of an input sequence but has not yet received all the characters in the sequence, it waits for more characters. If the timeout occurs before more characters are received, the received characters are sent to the screen, and the **Dynamic Screen** utility does not consider these characters as part of an input key sequence. Consider increasing the value of the **dsp** string if one or more of the keys to which the utility has to respond is actually a number of key combinations (for example, <Ctrl-Z> 1, <Ctrl-Z> 2, <Ctrl-Z> 3, and so on, for screen selection, or <Ctrl-Z> N, for new screen).

Examples

1. The following **dsinfo** entry describes a WYSE 60 terminal with three screens:

```
wy60|wyse60|wyse model 60,
   dsks=^A'^M|Shift-F1|,
   dsks=^Aa^M|Shift-F2|,
   dsks=^Ab^M|Shift-F3|,
   dskc=\200|Ctrl-F1|,
   dske=\201|Ctrl-F2|\Ew0\E+,
   dskl=\202|Ctrl-F3|,
   dsp=\Ew0|\E+,
   dsp=\Ew1|\E+,
   dsp=\Ew2|\E+,
```

The <Shift-F1> through <Shift-F3> key combinations are used for selecting screens 1 through 3. <Ctrl-F1> creates a new screen. <Ctrl-F2> sends the key sequence <Esc> w 0 <Esc> + to the screen. As a result, the terminal switches to window 0, the screen is cleared, and the **Dynamic Screen** utility ends. <Ctrl-F3> lists the keys and their functions. The three physical screens are displayed by sending the key sequences <Esc> w 0 , <Esc> w 1, and <Esc > w 2, respectively. Each time a physical screen is used for a new screen the <Esc> + key sequence is sent to the terminal to clear the screen.

2. The following **dsinfo** entry describes a WYSE 60 terminal with three screens, one of which is on a second computer communicating through the second serial port on the terminal. The **Dynamic Screen** utility must be run on both computers, with terminal type WY60-1 on the first computer and terminal type WY60-2 on the second computer (to do so specify the **-t** flag in the **dscreen** command).

```
wy60-1|wyse60-1|wyse model 60 - first
serial port
   dsks=^A‘^M|Shift-F1|,
   dsks=^Aa^M|Shift-F2|,
   dskb=^Ab^M|Shift-F3|\Ed#^Ab\r^T\Ee9,
   dskc=\200|Ctrl-F1|,
   dske=\201|Ctrl-F2|\Ed#\201^T\Ew0\E+,
   dskl=\202|Ctrl-F3|,
   dsp=\Ew0|\E+,dsp=\Ew1|\E+,
wy60-2|wyse60-2|wyse model 60 - second
serial port
   dskb=^A‘^M|Shift-F1|\Ed#^A‘\r^T\Ee8,
   dskb=^Aa^M|Shift-F2|\Ed#^Aa\r^T\Ee8,
   dsks=^Ab^M|Shift-F3|
   dskc=\200|Ctrl-F1|,
   dske=\201|Ctrl-F2|\Ed#\201^T\Ew0\E+,
   dskl=\202|Ctrl-F3|,
   dsp=\Ew2|\E+,
```

The first two key entries for terminal type WY60-1 are identical to the entry in example 1. The third key entry, of type `dskb`, specifies that input and output are blocked when the <Esc> d # <Ctrl-A> b <CR> <Ctrl-T> <Esc> e 9 key sequence is sent to the terminal. As a result, output is blocked, and the **Dynamic Screen** utility continues to scan input for key sequences but discards all other input. The <Esc> d # sequence puts the terminal in transparent print mode, which echoes all keystrokes up to <Ctrl-T> out the other serial port. The <Ctrtl-A> b <CR> key sequence is sent out to the other serial port, informing the **Dynamic Screen** utility on the second computer that it should activate the window associated with the <Shift-F3> key. The <Ctrl-T> key sequence takes the terminal out of transparent print mode, and the <Esc> e 9 key sequence informs the terminal to switch to the other serial port for data communications.

The other computer takes over and sends the <Esc> w 2 key sequence to switch to the third physical screen and then resumes normal communication.

The WY60-2 entry follows the same general pattern for the <Shift-F1> and <Shift-F2> key combinations, which switch to transparent print mode, send a function key string to the other computer, switch transparent print off, and switch to the other serial port.

The end key <Ctrl-F2> works the same for both computers. It sends the end key sequence to the other computer through the transparent print mechanism, switches the terminal to window 0, clears the screen, and exits.

Implementation Specifics

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

Files

/etc/dsinfo	Contains the terminal descriptions for the **Dynamic Screen** utility.

Related Information

The **dscreen** command.

dumpdates File

Purpose

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

Description

The **/etc/dumpdates** file holds filesystem backup information for the **backup** and **rdump** commands. The **dumpdates** file is maintained by using the **-u** option when performing file system backups. The following is the **dumpdates** data 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.
`id_incno`	The level number of the last backup.
`id_ddate`	The date of the incremental backup in system format.
MAXNAMLEN	The maximum value of this variable is 255.

Implementation Specifics

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

Files

/etc/dumpdates	Specifies the path name of the symbolic link to the **dumpdates** file.

Related Information

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

e789_ctbl File for HCON

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 Host Connection Program (HCON) in binary form.

Instances of the **e789_ctbl** file can also occur in user **$HOME** directories. The color definition table can be customized using the **hconutil** command. If the user issuing the **hconutil** command does not specify a name for the new table, the command names the **e789_ctbl** table and places it in the user **$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 Host Connection Program (HCON).

Files

/usr/lib/hcon/e789_ctbl	Specifies the path of the **e789_ctbl** file.

Related Information

The **hconutil** command, **chhcons** command, **mkhcons** command.

e789_ktbl File for HCON

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 Host Connection Program (HCON) in binary form.

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 **hconutil** command.

Instances of the **e789_ktbl** file can also occur in user **$HOME** directories. The keyboard definition table can be customized using the **hconutil** command. If the user issuing the **hconutil** command does not specify a name for the new table, the command names the **e789_ktbl** table and places it in the user **$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 Host Connection Program (HCON).

Files

/usr/lib/hcon/e789_ktbl	Specifies the path of the **e789_ktbl** file.

Related Information

The **hconutil** command, **chhcons** command, **mkhcons** command.

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*=*Value* 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.

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 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 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 **/etc/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 **/usr/lib/security/mkuser.default** file. The **chuser** command can change these attributes, and the **lsuser** command can display them. The **rmuser** command removes the entire record for a user.

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	file name

Examples

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

```
dhs:
   usrenv = "MAIL=/home/spool/mail/dhs,MAILCHECK=600"
   sysenv = "NAME=dhs@delos"
```

Implementation Specifics

This command is part of 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.
/usr/lib/security/mkuser.default	Contains the default values for user accounts.
/etc/security/lastlog	Contains last login information.

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.

environment File

Purpose

Sets up the user environment.

Description

The **/etc/environment** file contains variables specifying the basic environment for all processes. 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.

Environment variables are examined when a command starts running. The environment of a process is not changed by altering the **/etc/environment** file. Any processes that were started prior to the change to the **/etc/environment** file must be restarted if the change is to take effect for those processes. If the **TZ** variable is changed, the **cron** daemon must be restarted, because this variable is used to determine the current local time.

The following restrictions apply, when modifying the **environment** file:

- Ensure that newly created environment variables do not conflict with standard variables such as **MAIL**, **PS1**, **PS2**, and **IFS**.
- Ensure that the information in the **environment** file is in the *Name*=*Value* format. Unlike **profile** scripts, the **environment** file is not a shell script and does not accept data in any format other than the *Name*=*Value* format.

The Basic Environment

When you log in, the system sets environment variables from the **environment** file before reading your login profile, **.profile**.

The following variables make up the basic environment:

HOME The full path name of the user 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/environment** file at installation time.

NLSPATH The full path name for message catalogs. The default is:

/usr/lib/nls/msg/%L/%N:

/usr/lib/nls/msg/%L/%N.cat:

where **%L** is the value of the **LC_MESSAGES** category and **%N** is the catalog file name.

Note: See the **chlang** command for more information about changing message catalogs.

LC__FASTMSG If **LC_FASTMEG** is set to **false**, POSIX-compliant message handling is performed. If **LC__FASTMSG** is set to **true**, it specifies that default messages should be used for the C and POSIX locales and that **NLSPATH** is ignored. If this variable is set to anything other than **false** or **unset**, it is considered the same as being set to **true**. The default value is **LC__FASTMSG=true** in the **/etc/environment** file.

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

Note: All **setuid** and **setgid** programs will ignore the **LOCPATH** environment variable.

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/environment** file. The **TZ** environment variable has the following format (spaces inserted for readability):

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

The fields within the **TZ** environment variable are defined as follows:

std and **dst**

Designate the standard (**std**) and summer (**dst**) time zones. Only the **std** value along with the appropriate **offset** value is required. If the **dst** value is not specified, summer time does not apply. The values specified may be no less than three and no more than **TZNAME_MAX** bytes in length. The length of the variables corresponds to the `%Z` field of the **date** command; for **libc** and **libbsd**, **TZNAME_MAX** equals three characters. Any nonnumeric ASCII characters except the following may be entered into each field: a leading : (colon), a , (comma), a **-** (minus sign), a + (plus sign), or the ASCII null character.

Note: POSIX 1.0 reserves the leading : (colon) for an implementation-defined **TZ** specification. AIX disallows the leading colon, selecting **CUT0** and setting the `%Z` field to a null string.

An example of **std** and **dst** format is as follows:

```
EST5EDT
```

`EST`

Specifies Eastern U.S. standard time.

`5`

Specifies the offset, which is 5 hours behind Coordinated Universal Time (CUT).

`EDT` Specifies the corresponding summer time zone abbreviation.

Note: See "Time Zones" for a list of time zone names defined for the system.

offset

Denotes the value added to local time to equal Coordinated Universal Time (CUT). CUT is the international time standard that has largely replaced Greenwich Mean Time. The **offset** variable has the following format:

```
hh:mm:ss
```

The fields within the **offset** variable are defined as follows:

`hh` Specifies the **dst** offset in hours. This field is required. The `hh` value can range between the integers -12 and +11. A negative value indicates the time zone is east of the prime meridian; a positive value or no value indicates the time zone is west of the prime meridian.

`mm` Specifies the **dst** offset detailed to the minute. This field is optional. If the `mm` value is present, it must be specified between 0 and 59 and preceded by a : (colon).

`ss` Specifies the **dst** offset detailed to the second. The `ss` field is optional. If the `ss` value is present, it must be specified between 0 and 59 and preceded by a : (colon).

An **offset** variable must be specified with the **std** variable. An **offset** variable for the **dst** variable is optional. If no offset is specified with the **dst** variable, the system assumes that summer time is one hour ahead of standard time.

As an example of offset syntax, Zurich is one hour ahead of CUT, so its offset is -1. Newfoundland is 1.5 hours ahead of eastern U.S. standard time zones. Its syntax can be stated as any of the following: 3:30, 03:30, +3:30, or 3:30:00.

rule The **rule** variable indicates when to change to and back from summer time. The **rule** variable has the following format:

```
start/time,end/time
```

The fields within the **rule** variable are defined as follows:

`start` Specifies the change from standard to summer time.

`end` Specifies the return to standard time from summer time.

`time` Specifies when the time changes occur within the time zone. For example, if the `time` variable is encoded for 2 a.m. then the time changes when the time zone reaches 2 a.m. on the date specified in the `start` variable.

`/` Delimits the start date, end date, and time variables.

`,` (Comma) Delimits two date and time pairs.

The `start` and `end` variables support a syntax for Julian time (`J`) and a syntax for leap years (`M`):

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

In the `J` syntax, the `n` variable has the value of 1 through 365. Leap days are not counted. In the `M` syntax, `m` is the month, `n` the week, and `d` the day of the week starting from day 0 (Sunday).

The **rule** variable has the same format as the **offset** variable except no leading - (minus sign) or + (plus sign) is allowed. The default of the `start` variable is 02:00:00 (2 a.m.).

Note: The time zone offsets and time change points are interrelated and context-dependent. The **rule** variable's runtime execution semantics change as a function of the offsets. For example, if the summer time zone changes one hour, as in `CST6CDT5`, (the default 2 a.m.) summer time changes instantaneously from 2 a.m. to 3 a.m. CDT. The fall change is from 2 a.m. CDT to 1 a.m. CST. The respective changes for a time zone of `CST6CDT4` are 2 a.m. CST to 4 a.m. CDT and 2 a.m. CDT to 12 a.m. CST.

In an example of the **rule** variable, if the law changed so that the Central United States experienced summer time between Julian 129 and Julian 131, the **TZ** variable would be stated as follows:

```
TZ=CST6CDT5,J129,J131
```

In this example, the dates indicated are May 09 and May 11,1993, respectively. (Use the **date +%j** command to get the Julian date number.)

In another example, if the time changes were to occur at 2 a.m. CST and 19:30 CDT, respectively, the variables would be stated as follows:

```
TZ=CST6CDT5,J129,J131/19:30
```

In nonleap years, the fallback time change would be from 19:30 CDT to 18:30 CST on May 11 (1993).

For the leap year (`M`) syntax, the spring ahead date would be 2 May and the fallback date is 9 May. The variables are stated as follows:

```
TZ=CST6CDT5,M5.1.0,M5.2.0
```

Time Zones

The system defines the following time zones and time zone names:

Note: Coordinated Universal Time (CUT) is the international time standard.

Time Zones Defined on the System		
Name	**Time Zone**	**CUT Offset**
CUT0GDT	Coordinated Universal Time	CUT
GMT0BST	United Kingdom	CUT
AZOREST1AZOREDT	Azores, Cape Verde	CUT -1
FALKST2FALKDT	Falkland Islands	CUT -2
GRNLNDST3GRNLNDDT	Greenland, East Brazil	CUT -3
AST4ADT	Central Brazil	CUT -4
EST5EDT	Eastern United States, Colombia	CUT -5
CST6CDT	Central United States, Honduras	CUT -6
MST7MDT	Mountain United States	CUT -7
PST8PDT	Pacific United States, Yukon	CUT -8
AST9ADT	Alaska	CUT -9
HST10HDT	Hawaii, Aleutian Islands	CUT -10
BST11BDT	Bering Strait	CUT -11
NZST-12NZDT	New Zealand	CUT +12
MET-11METDT	Solomon Islands	CUT +11
EET-10EETDT	Eastern Australia	CUT +10
JST-9JSTDT	Japan	CUT +9
KORST-9KORDT	Korea	CUT +9
WAUST-8WAUDT	Western Australia	CUT +8
TAIST-8TAIDT	Taiwan	CUT +8
THAIST-7THAIDT	Thailand	CUT +7
TASHST-6TASHDT	Central Asia	CUT +6
PAKST-5PAKDT	Pakistan	CUT +5
WST-4WDT	Gorki, Central Asia, Oman	CUT +4
MEST-3MEDT	Turkey	CUT +3

SAUST-3SAUDT	Saudi Arabia	CUT +3
WET-2WET	Finland	CUT +2
USAST-2USADT	South Africa	CUT +2
NFT-1DFT	Norway	CUT +1

Implementation Specifics

This file is part of 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 specific user needs.
/etc/passwd	Specifies user IDs.

Related Information

The **at** command, **chlang** command, **env** command, **getty** command, **login** command, **sh** command.

errors File for BNU

Purpose

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

Description

The **/var/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.

Examples

The text of an error which might appear in the **errors** file is:

```
ASSERT ERROR (uucico) pid: 303 (7/18-8:25:09) SYSTAT OPEN FAIL /v
ar/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 **/var/spool/uucp/.Status** directory [`SYSTAT OPEN FAIL /var/spool/uucp/.Status/`]. To prevent this error from occurring again, you should make sure the 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/var/spool/uucp/.Admin directory	Contains the **errors** file and other BNU administrative files.
/var/spool/uucp/.Status/*SystemName*	Lists the last time a remote system was contacted and the minimum time until the next retry.
/var/spool/uucp/.Admin/errors	Specifies the path of the **errors** file.

Related Information

The **uudemon.cleanu** command.

The **uucico** daemon.

ethers File for NIS

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 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, new line, 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 (BOS) Runtime.

Files

/etc/ethers	Specifies the path of the **ethers** file.
/etc/hosts	Contains Internet addresses.

Related Information

The **/etc/hosts** file.

NFS Services in *AIX Version 4.3 System Management Guide: Communications and Networks*.

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.
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.

Audit Event Formatting Information

Format	Description
%A	Formatted output is similar to the **aclget** command.
%d	Formatted as a 32-bit signed decimal integer
%G	Formatted as a comma-separated list of group names or numerical identifiers.
%o	Formatted as 32-bit octal integer.
%P	Formatted output is similar to the **pclget** command.
%s	Formatted as a text string.
%T	Formatted as a text string giving include date and time with 6 significant digits for the seconds `DD Mmm YYYY HH:MM:SS:mmmuuu`).
%u	Formatted as a 32-bit unsigned integer.
%x	Formatted as a 32-bit hexidecimal integer.
%X	Formatted as a 32-bit hexidecimal integer with upper case letters.

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

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

```
auditpr:
  FILE_Open = printf "flags: %d mode: %o \
   fd: %d filename: %s"
  PROC_Create = printf "forked child process %d"
```

Implementation Specifics

This file is part of 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 **audit** command, **auditpr** command.

Setting Up Auditing in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

Auditing Overview and

Execute (X.*) Files for BNU

Purpose

Contains instructions for running commands that require the resources of a remote system.

Description

The execute (**X.***) files of the Basic Networking Utilities (BNU) 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:

/var/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:

User Line

U *UserName SystemName*	Specifies the login name of the user issuing the **uux** command and the name of the system that issued the command.

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 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 **/etc/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 **/var/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 /home/amy/out hera!/home/amy/out2 > ~/DF"
   ```

   The command in this example invokes the **uux** command to run a **diff** command on the local system, comparing the /home/amy/out file with the /home/amy/out2 file, 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 /var/spool/uucp/hera/X.zeusN212F execute file, which contains the following information:

   The user line identifies the user amy on the 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:

   ```
   U amy zeus
   # return status on failure
   Z
   # return address for status or input return
   R amy
   F /var/spool/uucp/hera/D.herale954fd out2
   O ~/DF zeus
   C diff /home/amy/out out2
   /var/spool/uucp/hera/D.herale954fd out2
   ```

   The output of the command is to be written to the public directory on the system zeus with the file name DF. (The ~ (tilde) is the shorthand way of specifying the public directory.) The final line is the command string that the user amy entered with the **uux** command.
2. The 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp/Permissions	Describes access permissions for remote systems.
/etc/uucp/Systems	Describes accessible remote systems.
/var/spool/uucp/*SystemName* directory	Contains BNU command, data, and execute files.
/var/spool/uucp/*SystemName***/C.***	Contains instructions for transfers.
/var/spool/uucp/.Xqtdir directory	Contains lists of commands that remote systems are permitted to execute.
/var/spool/uucppublic/* directory	Contains transferred files.

Related Information

The **diff** command, **uux** command.

The **uuxqt** daemon.

exports File for NFS

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** daemons.

> **Note:** You cannot export either a parent directory or a subdirectory of an exported directory within the same file system.

Entries in the file are formatted as follows:

Directory -*Option* [**,** *Option*] ...

These entries are defined as follows:

Directory Specifies the directory name.

Option Specifies optional characteristics for the directory being exported. You can enter more than one variable 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 host name or the network name. If a *rw* host name is not specified, the directory is exported with read-write permission to all.

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 in the **/etc/netgroup** database and then in the **/etc/hosts** database. The default value allows any machine to mount the given directory.

anon= *UID* If a request comes from a root user, use the user identification (*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.

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. To export to **netgroup** clients, enter:

```
/usr -access=clients
```

2. To export to the world, enter:

```
/usr/local
```

3. To export to only these systems, enter:

```
/usr2 -access=hermes:zip:tutorial
```

4. To give root access only to these systems, enter:

```
/usr/tps -root=hermes:zip
```

5. To convert client root users to guest UID=100, enter:

```
/usr/new -anon=100
```

6. To export read-only to everyone, enter:

```
/usr/bin -ro
```

7. To allow several options on one line, enter:

```
/usr/stuff -access=zip,anon=-3,ro
```

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System (BOS) 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.

List of NFS Files.

.fig File

Purpose

Contains a list of **F** file names.

Description

The **.fig** file is one of several intermediate files produced for each document by InfoCrafter. The **.fig** file is an ASCII file that contains a list of **F** file names created for the document. **F** files are files containing artwork.

Implementation Specifics

This file is part of the InfoCrafter product.

Files

.fig	Contains a list of **F** file names.

Related Information

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 **/etc/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 mount automatically 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 to build 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.

vfs Specifies the type of mount. For example, **vfs=nfs** specifies the virtual file system being mounted is an NFS file system.

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 *LVName* must be the full path name of the filesystem logging logical volume name to which log data is written as this file system is modified. This is only valid for journaled file systems.

Examples

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

Attention: 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

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

/home/joe/1:
         dev        = /home/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
```

Note: The asterisk (`*`) is the comment character used in the **/etc/filesystems** file.

Implementation Specifics

This file is part of Base Operating System 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 **backup** command, **df** command, **dodisk** command, **fsck** command, **mkfs** command, **mount** command, **restore** command, **umount** command.

The **filesys.h** file.

Files Overview.

Directory Overview and

Foreign File for BNU

Purpose

Logs contact attempts from unknown systems.

Description

The **/var/spool/uucp/.Admin/Foreign** file lists access attempts by unknown systems. The **/usr/sbin/uucp/remote.unknown** shell script appends an entry to the **Foreign** file each time a remote computer that is not listed in the local **/etc/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.

Examples

This 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 **/etc/uucp/Systems** file, attempted to log in 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/var/spool/uucp/.Admin/Foreign	Specifies the path of the **Foreign** file.
/etc/uucp/Permissions	Describes access permissions for remote systems.
/etc/uucp/Systems	Describes accessible remote systems.
/usr/sbin/uucp/remote.unknown	Records contacts from unknown systems in the **Foreign** file.
/var/spool/uucp/.Admin directory	Contains BNU administrative files.

Related Information

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

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

.forward File

Purpose

Automatically forwards mail as it is received.

Description

When mail is sent to a local user, the **sendmail** command checks for the **$HOME/.forward** file. The **$HOME/.forward** file can contain one or more addresses or aliases. If the file exists, the message is not sent to the user. The message is sent to the addresses or aliases in the **.forward** file. For example, if user `mickey's` **.forward** file on host `disney` contains:

```
donald@wonderful.world.disney
pluto
```

Copies of messages sent to `mickey` are forwarded to user `donald` on host `wonderful.world.disney`, and to `pluto` on the local system.

Notes:

1. The addresses listed in the **.forward** file can be a comma-separated list of addresses; for example:

   ```
   donald@wonderful.world.disney, pluto
   ```

2. Addresses can specify programs. The following example forwards a message to the **vacation** command:

   ```
   mickey, "|/usr/bin/vacation mickey"
   ```

   This example sends a message to user `mickey` and to the **vacation** program.

3. This file must be created by the user in the **$HOME** directory.

To stop forwarding mail, use the **rm** command to remove the **.forward** file from your home directory:

```
rm .forward
```

The **.forward** file is deleted. Incoming mail is delivered to the user's system mailbox.

Implementation Specifics

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

Files

$HOME/.forward	Specifies the path of the file.

Related Information

The **mail** command, **vacation** command.

/etc/group File

Purpose

Contains basic group attributes.

Description

The **/etc/group** file contains basic group attributes. This is an ASCII file that contains records for system groups. Each record appears on a single line and is the following format:

*Name***:***Password***:***ID***:***User1*,*User2*,*...***,***Usern*

You must separate each attribute with a colon. Records are separated by new-line characters. The attributes in a record have the following values:

Name	Specifies a group name that is unique on the system. The name is a string of 8 bytes or less. See the **mkgroup** command for information on the restrictions for naming groups.
Password	Not used. Group administrators are provided instead of group passwords. See the **/etc/security/group** file for more information.
ID	Specifies the group ID. The value is a unique decimal integer string.
User1,*User2*,*...***,***Usern*	Identifies a list of one or more users. Separate group member names with commas. Each user must already be defined in the local database configuration files.

Do not use a : (colon) in any of the attribute fields. For an example of a record, see the "Examples" section . Additional attributes are defined in the **/etc/security/group** file.

Note: Certain system-defined group and user names are required for proper installation and update of the system software. Exercise care before replacing the **/etc/group** file to ensure that no system-supplied groups or users are removed.

You should access the **/etc/group** file through the system commands and subroutines defined for this purpose. You can use the following commands to manage groups:

- **chgroup**
- **chgrpmem**
- **chuser**
- **lsgroup**
- **mkgroup**
- **mkuser**

- **rmgroup**

To change the *Name* parameter, you first use the **mkgroup** command to add a new entry. Then, you use the **rmgroup** command to remove the old group. To display all the attributes in the file, use the **lsgroup** command.

You can use the **chgroup**, **chgrpmem**, or **chuser** command to change all user and group attributes. The **mkuser** command adds a user whose primary group is defined in the **/usr/lib/security/mkuser.default** file and the **rmuser** command removes a user. Although you can change the group ID 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.

Examples

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

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

In this example, the *GroupID* parameter is `1` and the users are defined to be `shadow` and `cjf`.

Implementation Specifics

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

Files

/etc/group	Contains basic group attributes.
/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 **chgroup** command, **chgrpmem** command, **lsgroup** command, **mkgroup** command, **rmgroup** command, **setgroups** command, **setsenv** command.

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

/etc/security/group File

Purpose

Contains extended group attributes.

Description

The **/etc/security/group** file contains extended group attributes. This is an ASCII file that contains a stanza for each system group. 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 pair ends with a new-line character as does each stanza. You can have multiple default stanzas in the **/etc/security/group** file. A default stanza applies to all of the stanzas that follow, but does not apply to the stanzas preceding it.

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

adms Defines the group administrators. Administrators are 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.

dce_export Allows the DCE registry to overwrite the local group information with the DCE group information during a DCE export operation. Possible values are:

- **true** Local group information will be overwritten.
- **false** Local group information will not be overwritten.

For a typical stanza, see the "Examples" section .

You should access the **/etc/security/group** file through the system commands and subroutines defined for this purpose. You can use the following commands to manage groups:

- **mkgroup**
- **chgroup**
- **chgrpmem**
- **lsgroup**
- **rmgroup**

The **mkgroup** command adds new groups to the **/etc/group** file and the **/etc/security/group** file. Use this command to create an administrative group. You can also use the **mkgroup** to set the group administrator.

Use the **chgroup** command to change all the attributes. If you are an administrator of a 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. The **rmgroup** command removes the entry from both the **/etc/group** file and the **/etc/security/group** file.

To write programs 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	file name

Examples

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

```
finance:
          admin = false
          adms = cjf, scott, sah
```

Implementation Specifics

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

Files

/etc/security/group	Specifies 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.
/etc/security/lastlog	Contains last login 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.

image.data File

Purpose

Contains information on the image installed during the Base Operating System installation process.

Description

The **image.data** file contains information describing the image installed during the BOS installation process. This information includes the sizes, names, maps, and mount points of logical volumes and file systems in the root volume group. The **mkszfile** command generates the **image.data** file. It is not recommended that the user modify the file. Changing the value of one field without correctly modifying any related fields can result in a failed installation and a corrupted backup image. The only exception to this recommendation is the `SHRINK` field, which the user may modify to instruct the BOS installation routines to create the file systems as specified in the **image.data** file or to create the file systems only as large as is required to contain all the data in the file system.

The BOS installation process also takes input from the **image.data** file regarding defaults for the machine being installed. Any default values in the **image.data** file will override values obtained when the BOS installation queries the hardware topology and existing root volume group. The **image.data** file resides in the **/** directory.

Note: The **image.data** file replaces the **.fs.size** file used in release 3.2.

The **image.data** file is arranged in stanza format. Each stanza contains one or more fields. These stanzas include the following:

- image_data
- logical_volume_policy
- ils_data
- vg_data
- source_disk_data
- lv_data
- fs_data
- post_install_data
- post_restvg

image_data Stanza

`IMAGE_TYPE`	Identifies the format of the image. Examples include backup file format (bff) and tar format.
`DATE_TIME`	Contains the date and time that the image was taken.
`UNAME_INFO`	Identifies the system and system level data asociated with the image.
`PRODUCT_TAPE`	Specifies whether the image is a product image or a **mksysb** image. The possible field values are `yes` or `no`.
`USERVG_LIST`	Lists the user volume groups defined in the system.
`OSLEVEL`	Identifies the **version.release.maintenance.fix** level of the system at the time the image was taken

Note: The `PRODUCT_TAPE` and `USERVG_LIST` fields are only present for the ROOTVG volume group.

logical_volume_policy Stanza

`SHRINK`	Instructs BOS install routines to create the file systems as they are specified in the **image.data** file or create the smallest file systems required to contain all the data in the file system. The field value specified can be `yes` (shrink file systems) or `no` (use **image.data** file specifications).
`EXACT_FIT`	The field value specified can be `yes` or `no`. If `yes` is specified, the disk information listed in the source_disk_data stanza must match the actual disks found on the target machine during installation.

ils_data Stanza

`LANG`	Sets the language used by the BOS Install program.

vg_data Stanza

Notes:

1. The **image.data** file can contain only one vg_data stanza.
2. Starting with AIX 4.3.3, two new fields (BIGVG and TFACTOR) have been added to the **vg_data Stanza**.

`VGNAME`	Specifies the volume group name.
`PPSIZE`	Specifies the size of the physical partition for the volume group.
`VARYON`	Activates the volume group and all associated logical volumes so that the volume group is available for use. The field value can be `yes` or `no`.
`VG_SOURCE_DISK_LIST`	Lists the disks in the volume group.
`QUORUM`	If set to 1, indicates the volume group is to be automatically varied off after losing its quorum of physical volumes.
`CONC_CAPABLE`	Indicates a volume group is concurrent capable.
`CONC_AUTO`	Indicates a volume group is to be varried on automatically in concurrent mode.
`BIGVG`	Indicates a volume group is to be created as a big vg format volume group. This can accomodate up to 128 physical volumes and 512 logical volumes.
`TFACTOR`	Indicates a change in the limit of the number of physical partitions per physical volume.

source_disk_data Stanza

Note: The **image.data** file contains one source_disk_data stanza for each disk in the root volume group.

`PVID`	Specifies the 16 digit physical volume identifier for the disk.
`CONNECTION`	Specifies the combination of the **parent** and the **connwhere** attribute associated with a disk. The format for this field is: ***parent*** *attribute*//***connwhere*** *attribute*.
`LOCATION`	Specifies the locations of the disks in the root volume group.
`SIZE_MB`	Specifies the size, in MB, of the disks in the root volume group.
`HDISKNAME`	Specifies the names of the disks in the root volume group.

lv_data Stanza

Note: The **image.data** file contains one lv_data stanza for each logical volume created on the system.

`VOLUME_GROUP`	Specifies the logical volume group name. Volume group names must be unique systemwide and can range from 1 to 15 characters.
`LV_SOURCE_DISK_LIST`	Lists the disks in the logical volume.
`LV_IDENTIFIER`	Contains the identifier of the logical volume.

`LOGICAL_VOLUME`	Contains the name of the logical volume.
`PERMISSION`	Sets the access permissions. The field value can be `read/write` or `read` only.
`VG_STAT`	Indicates the state of the volume group. If the volume group is activated with the **varyonvg** command, the value of the `VG_STAT` field is either `active/complete` or `active/partial`. An `active/complete` field value indicates that all physical volumes are active, while an `active/partial` field value indicates that all physical volumes are not active. If the volume group is not activated with the **varonvg** command, the `VG_STAT` field value is inactive.
`TYPE`	Describes the logical volume type.
`MAX_LPS`	Sets the maximum number of logical partitions within the logical volume.
`COPIES`	Specifies the number of physical partitions created for each logical partition during the allocation process.
`LPS`	Specifies the number of logical partitions currently in the logical volume.
`STALE_PPs`	Specifies the number of physical partitions in the logical volume that are not current.
`INTER_POLICY`	Specifies the inter-physical allocation policy. The field value can be `minimum` or `maximum`.
`INTRA_POLICY`	Specifies the intra-physical allocation policy. The possible field values are either `middle`, `center`, or `edge`.
`MOUNT_POINT`	Specifies the file-system mount point for the logical volume, if applicable.
`MIRROR_WRITE_CONSISTENCY`	Specifies mirror-write consistency state. The field value can be `off` or `on`.
`LV_SEPARATE_PV`	Specifies a `yes`, `no`, or `super` field value for strict allocation. A `yes` value for strict allocation states that no copies for a logical partition are allocated on the same physical volume. A `no` value for strict allocation (nonstrict) states that at least one occurrence of two physical partitions belong to the same logical partition. A `super` value for strict allocation (super strictness) states that no partition from one mirror copy may reside on the same disk as another mirror copy.

LV_STATE	Describes the state of the logical volume. An Opened/stale value indicates the logical volume is open but contains physical partitions that are not current. An Open/syncd value indicates the logical volume is open and its physical partitions are current, or synchronized. A Closed value indicates the logical volume has not been opened.
WRITE_VERIFY	Specifies the field value of the write verify state as on or off.
PP_SIZE	Provides the size physical partition.
SCHED_POLICY	Specifies a sequential or parallel scheduling policy.
PP	Specifies the number of physical partitions currently in the logical volume.
BB_POLICY	Specifies the bad block relocation policy.
RELOCATABLE	Indicates whether the partitions can be relocated if a reorganization of partition allocation takes place. The field value can be yes or no.
UPPER_BOUND	Specifies the maximum number of physical volumes for allocation.
LABEL	Specifies the label field for the logical volume.
MAPFILE	Provides the full path name to a map file to be used in creating the logical volume.
LV_MIN_LPS	Specifies the minimum size of the logical volume to use when shrinking the logical volume.
STRIPE_WIDTH	Specifies the number of physical volumes being striped across.
STRIPE_SIZE	Specifies the number of bytes per stripe. The field value must be a power of two, between 4K and 128K; for example, 4K, 8K, 16K, 32K, 64K, or 128K.

fs_data Stanza

`FS_NAME`	Specifies the mount point of the file system.
`FS_SIZE`	Specifies the size, in 512-byte blocks, of the file system.
`FS_MIN_SIZE`	Specifies the minimum size required to contain the files of the file system. This size is used when the `SHRINK` field in the logical_volume_policy stanza has a field value of `yes`.
`FS_LV`	Provides the logical volume name. The name must contain the **/dev/** prefix. An example of an appropriate name is `/dev/hd4`.
`FS_FS`	Specifies the fragmentation size of the system. This value is optional.
`FS_NBPI`	Specifies the number of bytes per inode. This value is optional.
`FS_COMPRESS`	Designates whether the file system should be compressed or not. The field value can be `LZ`, which compresses the file system, or the `no` field value.
`FS_BF`	Enables the file system for files greater than 2 GB. The possible values are `true` or `false`.
`FS_AGSIZE`	Specifies the allocation group size. The possible values are `8`, `16`, `32`, or `64`. The allocation group size is specified in units of megabytes.

post_install_data Stanza

`BOSINST_FILE`	Provides the full path name of a file or command to execute after BOS install completes.

post_restvg Stanza

`RESTVG_FILE`	Specifies the full path name of the file or command to execute after the restvg process completes.

Note: The post_install_data stanza exists for the ROOTVG volume group and the post_restvg stanza is present for other volume groups.

Implementation Specifics

This file is part of System Backup and BOS Install Utilites.

Related Information

INed Files

Purpose

Contains programs and data used by the INed program.

Description

The **/usr/lib/INed** directory contains a number of files and subdirectories used internally by the INed program. The **/usr/lib/nls/msg/$LANG** directory contains files of translatable text. This directory also contains other files that are not used by INed.

In the following file names, **$LANG** is the value of the **lib/Language** environment variable, which indicates the national language currently being used.

bin	Directory containing programs called by the editor to perform various functions. Do not run these programs from the command line.
FATAL.LOG	Log of error messages the editor records when it encounters a system problem.
helpers	Directory containing programs called by the editor to help work on certain kinds of data. Files ending in **.x** or named **x** use the helper named **x.help**. Helpers typically supply the functions listed on the INed local menus.
forms	Directory containing forms used by the INed program. Files ending in **.x** or named **x** use the **x.ofm** form. The forms are binary files used directly by the editor in generating displays for structured files.
/usr/lib/nls/msg/$LANG/keys.map	File displayed when the Help command key (F1) is pressed and the keymap option is selected.
termcap	Directory containing the files used by the editor to read input from the terminals and write output to the terminals. The **def.trm** file is the readable structured file, and the **terms.bin** file is the compressed version.
/usr/lib/nls/msg/$LANG	Directory containing help message files and other files containing translated text used by the INed editor. This directory also contains other files not used by INed.

Implementation Specifics

These files are part of INed Editor Facilities.

Files

/usr/lib/INed directory	Contains files and subdirectories used by the INed program.
/usr/lib/nls/msg/$LANG directory	Contains files of translatable text.

Related Information

The **at** command, **cat** command, **del** command, **e** command, **format** command, **fill** command, **ghost** command, **history** command, **just** command, **keymaps** command, **newfile** command, **nl** command, **piobe** command, **prtty** command, **qprt** command, **readfile** command, **rmhist** command, **rpl** command, **sort** command, **stty** command, **tdigest** command, **trbsd** command, **untab** command, **versions** command.

Editors Overview in *AIX Version 4.3 INed Editor User's Guide* describes concepts and tasks specific to the INed editor.

.info File

Purpose

Stores configuration information used by the Network Install Manager (NIM).

Description

The **.info** file contains a series of Korn shell variable assignments used by NIM. The **.info** file is created by NIM for each client. During network boot, the **rc.boot** program uses several of these variables to control processing.

If a client is initialized by NIM, the **.info** file is copied into that client's **/etc** directory as the **/etc/niminfo** file. The **nimclient** command uses the **/etc/niminfo** file to communicate with the NIM master server.

> **Note:** The following variable groups are based upon the function of the variables that they contain. The **.info** file itself is not divided into categories.

Variables used directly by the rc.boot program

ROUTES
: Contains all the routing information the client needs in order to access any allocated NIM resource. This information is presented as a series of space-separated stanzas, each in the following format:

  *DestinationIPAddress***:***DestinationSubnet* **:***GatewayIPAddress*

SPOT
: Specifies the location of the shared product object tree (SPOT) to be used during the boot process. This variable contains the host and pathname of the client's SPOT in the following format:

  *HostName***:***SPOTDirectory*

RC_CONFIG
: Specifies the file name of the **rc.config** script to use.

NIM_HOSTS
: Provides information used to construct an /etc/hosts file for the client. The value is formatted as follows:

  *IPAddress***:***HostName IPAddress***:***HostName ...*

Variables used by any rc.config script

ROOT	Specifies the host and path name of the client's root directory in the following format: *HostName***:***RootDirectory*
MOUNTS	Contains a series of space-separated stanzas, each composed of a remote directory specification and the point where it should be mounted. The stanzas are in the following format: *HostName***:***RemoteDirectory***:***LocalDirectory*

Variables used by the nim commands

NIM_NAME	Designates the name of the client's NIM **machines** object.
NIM_CONFIGURATION	Specifies the client's NIM configuration machine type.
NIM_MASTER	Specifies the IP address of the NIM master server.
NIM_MASTER_PORT	Specifies the port number to use for client communications.
NIM_REGISTRATION_PORT	Specifies the port number to use for client registration.
NIM_MAX_RETRIES	Specifies the maximum number of retries for communication attempts with the **nimesis** daemon.
NIM_MAX_DELAY	Sets the amount of time to wait between retries for communication with the **nimesis** daemon.

Variables used by BOS Install

The following variables are used by NIM to control Base Operating System (BOS) installation operation:

NIM_BOSINST_DATA	Specifies the RAM file system path name to the **bosinst.data** file to be used. This variable has the following format: *Pathname*
NIM_BOS_IMAGE	Specifies the RAM file system path name to the BOS image.
NIM_CUSTOM	Specifies the path name of the customization script to execute after BOS installation.

Variables used by the rc.dd_boot Script

The **rc.dd** script uses the following variables to perform boot specific processing to create certain NIM resources.

DTLS_PAGING_SIZE Contains the paging-space size that you specify. If you have not set the paging space, the value is NULL and the **rc.dd_boot** script defaults to a paging space twice that of the client's RAM space.

DTLS_LOCAL_FS Contains a list of acronyms specifying the filesystems to be created locally on the client. The possible values are `tmp` and `home`.

Examples

The following is an example of a **.info** file:

```
#----------------Network Install
Manager---------
# warning - this file contains NIM configuration information
#         and should only be updated by NIM
export NIM_NAME=dua
export NIM_CONFIGURATION=standalone
export NIM_MASTER_HOSTNAME=satu
export NIM_MASTER_PORT=1058
export NIM_REGISTRATION_PORT=1059
export RC_CONFIG=rc.bos_inst
export SPOT=tiga:/usr
export NIM_CUSTOM=/tmp/dua.script
export NIM_BOS_IMAGE=/SPOT
export NIM_BOS_FORMAT=master
export NIM_HOSTS=" 130.35.130.1:satu 130.35.130.3:tiga "
export MOUNTS=" tiga:/export/logs/dua:/var/adm/ras:dir
tiga:/export/nim/simages
:/SPOT/usr/sys/inst.images:dir
satu:/export/nim/scripts/dua.script:tmp/dua.script:file "
```

Related Information

The **lsnim** command, **nim** command, **nimclient** command, **nimconfig** command, **niminit**

inittab File

Purpose

Controls the initialization process.

Description

The **/etc/inittab** file supplies the script to the **init** command's role as a general process dispatcher. The process that constitutes the majority of the **init** command's process dispatching activities is the **/etc/getty** line process, which initiates individual terminal lines. Other processes typically dispatched by the **init** command are daemons and the shell.

The **/etc/inittab** file is composed of entries that are position-dependent and have the following format:

`Identifier:RunLevel:Action:Command`

Note: The colon character (:) is used as a delimiter as well as a comment character. To comment out an **inittab** entry, add **:** at the beginning of the entry. For example:

```
:Identifier:RunLevel:Action:Command
```

Each entry is delimited by a newline character. A backslash (\) preceding a newline character indicates the continuation of an entry. There are no limits (other than maximum entry size) on the number of entries in the **/etc/inittab** file. The maximum entry size is 1024 characters. The entry fields are:

Identifier
A string (one or more than one character) that uniquely identifies an object.

RunLevel
The run level in which this entry can be processed. Run levels effectively correspond to a configuration of processes in the system. Each process started by the **init** command is assigned one or more run levels in which it can exist. Run levels are represented by the numbers 0 through 9. For example, if the system is in run level 1, only those entries with a 1 in the *runlevel* field are started. When you request the **init** command to change run levels, all processes without an entry in the *runlevel* field for the target run level receive a warning signal (**SIGTERM**). There is a 20-second grace period before processes are forcibly terminated by the kill signal (**SIGKILL**). The *runlevel* field can define multiple run levels for a process by selecting more than one run level in any combination from 0 through 9. If no run level is specified, the process is assumed to be valid at all run levels.

There are three other values that appear in the *runlevel* field, even though they are not true run levels: **a**, **b**, and **c**. Entries that have these characters in the *runlevel* field are processed only when the **telinit** command requests them to be run (regardless of the current run level of the system). They differ from run levels in that the **init** command can never enter run level **a**, **b**, or **c**. Also, a request for the execution of any of these processes does not change the current run level. Furthermore, a process started by an **a**, **b**, or **c** command is not killed when the **init** command changes levels. They are only killed if their line in the **/etc/inittab** file is marked off in the *action* field, their line is deleted entirely from **/etc/inittab**, or the **init** command goes into single-user mode.

Action Tells the **init** command how to treat the process specified in the *process* field. The following actions are recognized by the **init** command:

respawn If the process does not exist, start the process. Do not wait for its termination (continue scanning the **/etc/inittab** file). Restart the process when it dies. If the process exists, do nothing and continue scanning the **/etc/inittab** file.

wait When the **init** command enters the run level that matches the entry's run level, start the process and wait for its termination. All subsequent reads of the **/etc/inittab** file while the **init** command is in the same run level will cause the **init** command to ignore this entry.

once When the **init** command enters a run level that matches the entry's run level, start the process, and do not wait for its termination. When it dies, do not restart the process. When the system enters a new run level, and the process is still running from a previous run level change, the program will not be restarted. All subsequent reads of the **/etc/inittab** file while the **init** command is in the same run level will cause the **init** command to ignore this entry.

boot Process the entry only during system boot, which is when the **init** command reads the **/etc/inittab** file during system startup. Start the process, do not wait for its termination, and when it dies, do not restart the process. In order for the instruction to be meaningful, the run level should be the default or it must match the **init** command's run level at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait Process the entry the first time that the **init** command goes from single-user to multi-user state after the system is booted. Start the process, wait for its termination, and when it dies, do not restart the process. If the **initdefault** is 2, run the process right after boot.

powerfail Execute the process associated with this entry only when the **init** command receives a power fail signal (**SIGPWR**).

powerwait Execute the process associated with this entry only when the **init** command receives a power fail signal (**SIGTERM**), and wait until it terminates before continuing to process the **/etc/inittab** file.

off If the process associated with this entry is currently running, send the warning signal (**SIGTERM**), and wait 20 seconds before terminating the process with the kill signal (**SIGKILL**). If the process is not running, ignore this entry.

ondemand Functionally identical to **respawn**, except this action applies to the **a**, **b**, or **c** values, not to run levels.

initdefault An entry with this action is only scanned when the **init** command is initially invoked. The **init** command uses this entry, if it exists, to determine which run level to enter initially. It does this by taking the highest run level specified in the *runlevel* field and using that as its initial state. If the *runlevel* field is empty, this is interpreted as 0123456789; therefore, the **init** command enters run level 9. Additionally, if the **init** command does not find an **initdefault** entry in the **/etc/inittab** file, it requests an initial run level from the user at boot time.

sysinit Entries of this type are executed before the **init** command tries to access the console before login. It is expected that this entry will only be used to initialize devices on which the **init** command might try to ask the run level question. These entries are executed and waited for before continuing.

Command	A shell command to execute. The entire *command* field is prefixed with `exec` and passed to a forked `sh` as `sh -c exec command`. Any legal `sh` syntax can appear in this field. Comments can be inserted with the # comment syntax. The **getty** command writes over the output of any commands that appear before it in the **inittab** file. To record the output of these commands to the boot log, pipe their output to the **alog -tboot** command. The stdin, stdout and stdferr file descriptors may not be available while **init** is processing **inittab** entries. Any entries writing to stdout or stderr may not work predictably unless they redirect their output to a file or to **/dev/console**.

The following commands are the only supported method for modifying the records in the **/etc/inittab** file:

chitab Changes records in the **/etc/inittab** file.

lsitab Lists records in the **/etc/inittab** file.

mkitab Adds records to the **/etc/inittab** file.

rmitab Removes records from the **/etc/inittab** file.

Examples

1. To start the `ident` process at all run levels, enter:

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

2. To start the `ident` process only at run level 2, enter:

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

3. To disable run levels 0, 3, 6-9 for the `ident` process, enter:

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

4. To start the **rc** command at run level 2 and send its output to the boot log, enter:

   ```
   rc:2:wait:/etc/rc 2>&1 | alog -tboot >
   /dev/console
   ```

Implementation Specifics

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

Files

/etc/inittab Specifies the path of the **inittab** file.

/usr/sbin/getty Indicates terminal lines.

Related Information

The **chitab**command, **init** command, **lsitab** command, **mkitab** command, **rmitab** command, **telinit** command.

irs.conf File

Purpose

Specifies the ordering of certain name resolution services.

Description

The **/etc/irs.conf** file is used to specify the ordering mechanism lookups for network related data by the resolver libraries of AIX. Resolution of hostname, networks, services, protocols, and netgroups can be achieved by using the subroutines listed in the following table.

hostname	**gethostbyname**, **gethostaddr**, **gethostent**
networks	**getnetbyname**, **getnetbyaddr**, **getnetent**
services	**getservbyname**, **getservbyaddr**, **getservent**
protocols	**getprotobyname**, **getprotobyaddr**, **getprotoent**
netgroups	**getnetgrent**

Because these routines are commonly used in many TCP/IP applications, using the **irs.conf** file can control the directions of the queries made by these applications as well.

The default order for lookup mechanisms for hosts is:

1. Domain Name Server (DNS)
2. Network Information Service (NIS), if active
3. local

The default order for lookup mechanisms for networks is:

1. DNS
2. NIS, if active
3. local.

The default order for lookup mechanisms for other maps is:

1. NIS, if active
2. local.

The default order can be overwritten by creating the configuration file, **/etc/irs.conf**, and specifying the desired ordering.

The settings in the configuration file **/etc/netsvc.conf** override the settings in the **/etc/irs.conf** file. The environment variable *NSORDER* overrides the settings in the **/etc/irs.conf** and the **/etc/netsvc.conf** files.

To use DNS to get information concerning netgroups, protocols, and services you must create and use a Hesiod DNS Zone.

<*map*> <*mechanism*> [<*option*>]

Available maps:

services	Lists the port numbers, transport protocols, and names of well known services.
protocol	Retrieves offical names and protocol numbers of protocol aliases.
hosts	Defines the mappings between host names and their IP addresses.
networks	Retrieves network names and addresses.
netgroup	Retrieves groups of hosts, networks, and users in these group.

Available mechanisms:

local	Examine local configuration files (**/etc/hosts**, **/etc/protocols**, **/etc/services**, **/etc/netgroup**, and **/etc/networks** files).
dns	Get information from DNS. The **/etc/resolv.conf** file must be configured for this mechanism.
nis	Get information from NIS. The NIS client must be active on the system to use this mechanism.
local4	Same as **local** but only looks for IPv4 addresses.
local6	Same as **local** but only looks IPv6 addresses.
dns4	Same as **dns** but only queries for A records (IPv4 addresses).
dns6	Same as **dns** but only queries for AAAA records (IPv6 addresses).
nis4	Same as **nis** but only looks for IPv4 addresses.
nis6	Same as **nis** but only looks for IPv6 addresses.

Available options:

continue	If not found in [*mechanism*], then continue to the next line (which should list another mechanism for the same map).
merge	Answers from all merged [*mechanisms*] into one response.

Examples:

An example of how to use [**continue**]:

Contents of **/etc/irs.conf** file:

```
hosts local continue
hosts dns continue
hosts nis
```

Contents of **/etc/hosts** file:

```
1.2.3.4 host4
1.2.3.5 host5
1.2.3.6 host6
```

Contents of DNS:

```
1.2.3.2 host2
1.2.3.3 host3
```

Contents of NIS:

```
1.2.3.1 host1
```

Function Call: **gethostbyname**(*host1*)

Result: returns `1.2.3.1`

Reason: The [**continue**] option will inform the resolver to first look for the information within the first mechanism, and if the requested information is not found, then continue to the next mechanism indicated in the **/etc/irs.conf** file. In this case, the resolver will first look at the local **/etc/hosts** file for *host1* but will not find it. Next, it will look at the DNS, but again, will not find it. Finally, it will look at the NIS, find the address, and return `1.2.3.1`.

An example of how to use [**merge**]:

Contents of **/etc/irc.conf** file:

```
hosts local merge
hosts dns
```

Contents of **/etc/hosts** file:

```
1.1.1.1 hostname
```

Information in DNS:

```
1.1.1.2 hostname
```

Function Call: **gethostbyname**(*hostname*)

Result: returns `1.1.1.1 1.1.1.2`

Reason: The **merge** option indicates that the resolver should take all the answers from the selected mechanisms and merge the answers together into one reply.

Examples of the **/etc/irs.conf** file:

The following examples show how the **irs.conf** file should look to control the query order of different records per *mechanism* and *map*. These examples are not an exhaustive list; any combination of *mechanism* and *map* can be used.

To use only the local **/etc/hosts** file for hostname resolution, enter:

```
hosts local
```

To use only the DNS for hostname resolution and NIS for protocols resolution, enter:

```
hosts dns
protocols nis
```

To use only NIS for hostname addresses and netgroup resolution and the local files for services and networks, enter:

```
hosts nis
services local
netgroup nis
networks local
```

To try to resolve host names from the local **/etc/hosts** file, then if they are not found, try to resolve from DNS, then NIS, enter:

```
hosts local continue
hosts dns continue
hosts nis continue
```

To try to resolve host names from the local **/etc/hosts** file, merge any information found with any DNS information found, and then merge this information, if any, to any NIS information found. If no information is found, none is returned. If information is found in any source, it will return as merged information from all of the available sources. Enter:

```
hosts local merge
hosts dns merge
hosts nis
```

To examine the local **/etc/services** file for port numbers before going to NIS. This can speed up the request since going to the NIS for information is much slower than going to the local file. If the information is not found in the local file, NIS will be searched. Enter:

```
services local continue
services nis
```

To look for IPv4 network addresses only from DNS and IPv6 host addresses only from the local file, enter:

```
networks dns4
hosts local6
```

Files

/etc/hosts	Contains the Internet Protocol (IP) name and addresses of hosts on the local network.
/etc/protocols	Contains offical names and protocol numbers of protocol aliases.
/etc/services	Contains lists of the port numbers, transport protocols, and names of well known services.
/etc/netgroup	Contains a list of groups of hosts, networks, and users in these groups.
/etc/networks	Contains a list of network names and addresses.
/etc/resolv.conf	Contains Domain Name Protocol (DOMAIN) name-server information for local resolver routines.
/etc/netsvc.conf	Specifies the ordering of certain name resolution services.

Related Information

The **ypbind** daemon.

The **gethostbyname**, **gethostbyaddr**, and **gethostent** subroutines for hostnames.

The **getnetbyname**, **getnetbyaddr**, **getnetent** subroutines for networks.

The **getservbyname**, **getservbyport**, **getservent** subroutines for services.

The **getprotobyname**, **getprotobynumber**, **getprotoent** subroutines for protocols.

The **getnetgrent** subroutine for netgroups.

ispaths File

Purpose

Defines the location of all databases in a library.

Description

The **ispaths** file contains a block of information (a stanza) for each database in a library. A library consists of up to 63 standalone or cross-linked databases. The **ispaths** file for the default InfoExplorer database library resides in the **/usr/lpp/info/data** directory. The **ispaths** files for other public libraries may reside in the **/usr/lpp/info/data/***LibraryName* directory, and contain a stanza of information for each database in the library.

Each stanza must have the following format:

Line	Explanation of Content
id *DatabaseNumber*	Represents the number of the database. This number can be between 0 and 1462, with a maximum of 1563 databases in a library. (Database number 1563 is reserved for the InfoExplorer help database.) **Note:** The order of databases in the **ispaths** file must match the order of databases in the **dbnames** file used during the build process.
primnav TRUE	(Optional.) Indicates whether the database contains any of the primary navigation articles. The **primnav** line can be set to **TRUE** for only one database in the library. Omit this line unless its value is **TRUE**.
browseTRUE	(Optional.) Indicates whether the entire library is browse enabled with the browse button displayed in the reading window. Omit this line if its value is not **TRUE**.
glossary TRUE	(Optional.) Indicates whether the database contains glossary entries. The **glossary** line can be set to **TRUE** for only one database in the library. Omit this line unless its value is **TRUE**.
name *Database*	Specifies the name of the database.
title *DatabaseTitle*	Specifies the title that InfoExplorer assigns the database. This title is displayed in the search results window (the Match Lists window) and the Database selection window helps users narrow their searches.
key *DatabasePath***/***DatabaseName***.key**	Specifies the full path name of the database **.key** file.
rom*DatabasePath* /*DatabaseName***.rom**	Specifies the full path name of the database **.rom** file.

The optional field **browse** can be specified in any of the stanzas, and its value will be applied to the entire library. The **browse** field does not need to be specified in each stanza for each library that has browse capability.

Examples

The following is an example of an **ispaths** file for a sample database.

The **isprime** file for this database specifies these primary navigation articles:

- Commands
- System Calls
- Subroutines
- Special Files
- File Formats
- List of Tasks

- List of Books
- Education

All the top-level lists reside in the navigation database.

```
#########################################
#         info Navigation Database
#########################################
id        0
primenav  TRUE
browse    TRUE
name      nav
title     Navigation
key       /usr/lpp/info/%L/nav/nav.key
rom       /usr/lpp/info/%L/nav/nav.rom


#########################################
#       info System Calls Database
#########################################
id      1
name    calls
title   System Calls
key     /usr/lpp/info/%L/calls/calls.key
rom     /usr/lpp/info/%L/calls/calls.rom


#########################################
#       info Subroutines Database
#########################################
id      2
name    subs
title   Subroutines
key     /usr/lpp/info/%L/subs/subs.key
rom     /usr/lpp/info/%L/subs/subs.rom


#########################################
#       info Special Files Database
#########################################
id      3
name    file
title   Special Files
key     /usr/lpp/info/%L/file/file.key
rom     /usr/lpp/info/%L/file/file.rom


#########################################
#       info File Formats Database
#########################################
id      4
name    fls
title   File Formats
key     /usr/lpp/info/%L/fls/fls.key
rom     /usr/lpp/info/%L/fls/fls.rom


#########################################
#       info Commands Database
#########################################
id      5
```

```
name    cmds
title   Commands
key     /usr/lpp/info/%L/cmds/cmds.key
rom     /usr/lpp/info/%L/cmds/cmds.rom


#########################################
#       info Book Contents Database
#########################################
id      6
name    books
title   Content Lists
key     /usr/lpp/info/%L/books/books.key
rom     /usr/lpp/info/%L/books/books.rom


#########################################
#       info Education Database
#########################################
id      7
name    educ
title   Education
key     /usr/lpp/info/%L/educ/educ.key
rom     /usr/lpp/info/%L/educ/educ.rom
```

Implementation Specifics

This file is part of the InfoExplorer product.

Files

/usr/lpp/info/data/ispaths	Contains the **ispaths** file for the AIX library.
/usr/lpp/info/data/*LibraryName***/ispaths**	Contains the **ispaths** file for the *LibraryName* library.
/usr/lpp/info/data/*LibraryName***/isprime**	Contains the names and numbers of button labels for the primary navigation articles in *LibraryName*.

Related Information

The **isprime** file.

isprime File

Purpose

Specifies the labels for links to primary navigation articles.

Description

The **isprime** file specifies labels for buttons located at the bottom of a navigation window in the graphics version of InfoExplorer, or the Display menu options in the ASCII version of InfoExplorer. These button labels or menu options serve as links to the primary navigation articles. Labels for up to eight primary navigation articles can be defined in the **isprime** file. The text string that serves as the label or options can consist of any alphanumeric combination, including spaces.

> **Note:** You must create an **isprime** file for each library that is built. The **isprime** file must reside in the **/usr/lpp/info/data/***LibraryName* directory if the library ia a public library. For private libraries, the **isprime** file must reside in the same directory the **ispaths** file does.

The format for the **isprime** file is as follows:

1 *TextForFirstLink*
2 *TextForSecondLink*
3 *TextForThirdLink*
4 *TextForFourthLink*
5 *TextForFifthLink*
6 *TextForSixthLink*
7 *TextForSeventhLink*
8 *TextForEighthLink*

Examples

An **isprime** file for a sample database might look as follows:

```
1 Commands
2 System Calls
3 Subroutines
4 Special Files
5 File Formats
6 List of Tasks
7 List of Books
8 Education
```

Implementation Specifics

This file is part of the InfoCrafter product.

Files

/usr/lpp/info/data/*LibraryName***/isprime**	Contains labels for links to primary navigation articles.

Related Information

.kshrc File

Purpose

Contains a shell script that customizes the Korn-shell environment.

Description

The **$HOME/.kshrc** file is a shell script that customizes the Korn-shell environment. This **.kshrc** script often contains a list of environment variables, command aliases, and function definitions that customize the Korn-shell environment.

Each time you start a new instance of the Korn shell, the **ksh** command examines the value of the **ENV** environment variable set in the **$HOME/.profile** file. If the **ENV** environment variable contains the name of an existing, readable file, the **ksh** command runs this file as a shell script. By convention, this file is named **$HOME/.kshrc.** You can use another name, but you must set the **ENV** environment variable to point to it.

Examples

The following is a sample of a **.kshrc** script on one specific system. The contents of your **.kshrc** file can be significantly different.

```
# @(#).kshrc 1.0

# Base Korn Shell environment

# Approach:

#       shell             initializations go in ~/.kshrc
#       user              initializations go in ~/.profile
#       host / all_user  initializations go in /etc/profile
#       hard / software  initializations go in /etc/environment

# DEBUG=y       # uncomment to report

[ "$DEBUG" ] && echo "Entering .kshrc"

set -o allexport

# options for all shells --------------------------------

# LIBPATH must be here because ksh is setuid, and LIBPATH is
# cleared when setuid programs are started, due to security hole.

LIBPATH=.:/local/lib:/lib:/usr/lib

# options for interactive shells follow-------------------------
```

```
TTY=$(tty|cut -f3-4 -d/)
HISTFILE=$HOME/.sh_hist$(echo ${TTY} | tr -d '/')
PWD=$(pwd)
PS1='
${LOGNAME}@${HOSTNAME} on ${TTY}
[${PWD}] '

# aliases

[ "$DEBUG" ] && echo "Setting aliases"

alias man="/afs/austin/local/bin/man -e less"
alias pg="pg -n -p':Page %d: '"
alias more="pg -n -p':Page %d: '"
alias cls="tput clear"
alias li="li -v"    # put <>and[] around executables and
directories
alias sane="stty sane"
alias rsz='eval $(resize)'

# mail check

if [ -s "$MAIL" ]       # This is at Shell startup.  In
then echo"$MAILMSG"     # normal operation, the Shell checks
fi                      # periodically.

# aixterm window title

[[ "$TERM" = "aixterm" ]] && echo
"\033]0;$USER@${HOSTNAME%t1}\007"

# functions

[ "$DEBUG" ] && echo "Setting functions"

function pid { ps -e | grep $@ | cut -d" " -f1; }

function df {
  /bin/df $* | grep -v afs;
  echo "\nAFS:";
  /usr/afs/bin/fs listquota /afs;
}

function term {
  if [ $# -eq 1 ]
  then
    echo $TERM
    TERM=$1
    export TERM
  fi
  echo $TERM
}

function back {

  cd $OLDPWD
  echo $CWD $OLDPWD
}
```

```
[ "$DEBUG" ] && echo "Exiting .kshrc"

set +o allexport
```

Implementation Specifics

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

Files

/etc/environment	Contains system-wide environment variable definitions.
/etc/profile	Contains system-wide environment customization.
$HOME/.kshrc	Sets the user environment for each start of the Korn shell.
$HOME/.profile	Contains user-specific logon initialization.

Related Information

The **ksh** command.

limits File

Purpose

Defines process resource limits for users.

Description

> **Note:** Changing the limit does not affect those processes that started by **init**, or alternatively, **ulimits** are only used by those processes that go through the login processes.

The **/etc/security/limits** file defines process resource limits for users. This file is an ASCII file that contains stanzas that specify the process resource limits for each user. These limits are set by individual attributes within a stanza.

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. If you do not define an attribute for a user, the system applies default values.

If the hard values are not explicitly defined in the **/etc/security/limits** file but the soft values are, the system substitutes the following values for the hard limits:

Resource	**Hard Value**
Core Size	`unlimited`
CPU Time	`cpu`
Data Size	`unlimited`
File Size	`fsize`
Memory Size	`unlimited`
Stack Size	`unlimited`
File Descriptors	`unlimited`

> **Note:** Use a value of -1 to set a resource to `unlimited`.

If the hard values are explicitly defined but the soft values are not, the system sets the soft values equal to the hard values.

You can set the following limits on a user:

fsize	Identifies the soft limit for the largest file a user’s process can create or extend.
core	Specifies the soft limit for the largest core file a user’s process can create.
cpu	Sets the soft limit for the largest amount of system unit time (in seconds) that a user’s process can use.
data	Identifies the soft limit for the largest process data segment for a user’s process.
stack	Specifies the soft limit for the largest process stack segment for a user’s process.
rss	Sets the soft limit for the largest amount of physical memory a user’s process can allocate. This limit is not enforced by the system.
nofiles	Sets the soft limit for the number of file descriptors a user process may have open at one time.
core_hard	Specifies the largest core file a user’s process can create.
cpu_hard	Sets the largest amount of system unit time (in seconds) that a user’s process can use.
data_hard	Identifies the largest process data segment for a user’s process.
fsize_hard	Identifies the largest file a user’s process can create or extend.
rss_hard	Sets the largest amount of physical memory a user’s process can allocate. This limit is not enforced by the system.
stack_hard	Specifies the largest process stack segment for a user’s process.
nofiles_hard	Sets the soft limit for the number of file descriptors a user process may have open at one time.

Except for the **cpu** attribute, each attribute must be a decimal integer string representing the number of 512-byte blocks allotted to the user. The **cpu** attribute is a decimal integer string representing the amount of system unit time in seconds. For an example of a **limits** file stanza, see the "Examples" section .

When you create a user with the **mkuser** command, the system adds a stanza for the user to the **limits** file. Once the stanza exists, you can use the **chuser** command to change the user’s limits. To display the current limits for a user, use the **lsuser** command. To remove users and their stanzas, use the **rmuser** command.

> **Note:** 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	file name

Examples

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

```
dhs:
   fsize = 8192
   core = 4096
   cpu = 3600
   data = 1272
   stack = 1024
   rss = 1024
   nofiles = 2000
```

Implementation Specifics

This command is part of 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.
/usr/lib/security/mkuser.default	Contains the default values for user accounts.
/etc/security/lastlog	Contains last login information.

Related Information

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

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

File and System Security Overview in *AIX Version 4.3 System User's Guide: Operating System and Devices*.

login.cfg File

Purpose

Contains configuration information for login and user authentication.

Description

The **/etc/security/login.cfg** file is an ASCII file that contains stanzas of configuration information for login 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 ends with a new-line character, and each stanza ends 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.

Port Stanzas

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

herald	Defines the login message printed when the **getty** process opens the port. The default herald is the `login` prompt. The value is a character string.
herald2	Defines the login message printed after a failed login attempt. The default herald is the `login` prompt. The value is a character string.
logindelay	Defines the delay factor (in seconds) between unsuccessful login attempts. The value is a decimal integer string. The default value is 0, indicating no delay between unsuccessful login attempts.
logindisable	Defines the number of unsuccessful login attempts allowed before the port is locked. The value is a decimal integer string. The default value is 0, indicating that the port cannot lock as a result of unsuccessful login attempts.
logininterval	Defines the time interval (in seconds) in which the specified unsuccessful login attempts must occur before the port is locked. The value is a decimal integer string. The default value is 0.
loginreenable	Defines the time interval (in minutes) a port is unlocked after a system lock. The value is a decimal integer string. The default value is 0, indicating that the port is not automatically unlocked.

logintimes Specifies the times, days, or both the user is allowed to access the system. The value is a comma-separated list of entries of the following form:

```
[!]:time-time
   -or-
[!]day[-day][:time-time]
   -or-
[!]date[-date][:time-time]
```

The *day* variable must be one digit between 0 and 6 that represents one of the days of the week. A 0 (zero) indicates Sunday and a 6 indicates Saturday.

The *time* variable is 24-hour military time (1700 is 5:00 p.m.). Leading zeroes are required. For example, you must enter `0800,` not `800`. The *time* variable must be four characters in length, and there must be a leading colon (:). An entry consisting of only a time specification applies to every day. The start hour of a time value must be less than the end hour.

The *date* variable is a four digit string in the form *mmdd*. *mm* represents the calendar month and *dd* represents the day number. For example `0001` represents January 1. *dd* may be `00` to indicate the entire month, if the entry is not a range, or indicating the first or last day of the month depending on whether it appears as part of the start or end of a range. For example, `0000` indicates the entire month of January. `0600` indicates the entire month of June. `0311-0500` indicates April 11 through the last day of June.

Entries in this list specify times that a user is allowed or denied access to the system. Entries not preceded by an exclamation point (`!`) allow access and are called ALLOW entries. Entries prefixed with an exclamation point (`!`) deny access to the system and are called DENY entries. The `!` operator applies to only one entry, not the whole restriction list. It must appear at the beginning of an entry.

sak_enabled Defines whether the secure attention key (SAK) is enabled for the port. The SAK key is the Ctrl-X, Ctrl-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.

The **sak_enabled** stanza can also be modified to close a potential security exposure that exists when tty login devices are writable by others; for example, when the tty mode is 0622. If the **sak_enabled** stanza is set to True, the tty mode is set to a more restrictive 0600 at login. If the **sak_enabled** stanza is set to False (or absent), the tty mode is set to 0622.

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.

Synonyms are not associative. For example, if you specify `synonym=/dev/tty0` in the stanza for the **/dev/console** path name, then the **/dev/tty0** path name is a synonym for the **/dev/console** path name. However, the **/dev/console** path name is not a synonym for the **/dev/tty0** path name unless you specify `synonym=/dev/console` in the stanza for the **/dev/tty0** path name.

Authentication Method Stanzas

These stanzas define the authentication methods for users 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 Stanzas

User-configuration stanzas provide configuration information for programs that change user attributes. There is one user-configuration stanza: **usw**.

Note: Password restrictions have no effect if you are on a network using Network Information Services (NIS). See "Network Information Service (NIS) Overview for System Management" in *AIX Version 4.3 System Management Guide: Communications and Networks* for a description of NIS.

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

logintimeout	Defines the time (in seconds) the user is given to type the password. The value is a decimal integer string. The default is a value of 60.
maxlogins	Defines the maximum number of simultaneous logins to the system. The format is a decimal integer string. The default value varies depending on the specific machine license. A value of 0 indicates no limit on simultaneous login attempts. **Note:** Login sessions include rlogins and telnets; these are counted against the maximum allowable number of simultaneous logins by the **maxlogins** attribute.
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 **/usr/bin/sh**, **/usr/bin/bsh**, **/usr/bin/csh**, **/usr/bin/ksh**, or **/usr/bin/tsh**.

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	File name

Examples

1. A typical **authentication_method** stanza looks like the following:

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

2. A typical **port** stanza looks like the following:

```
/dev/tty0:
  sak_enabled = true
  herald = "login to tty0:"
```

Implementation Specifics

This command is part of 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.
/etc/security/lastlog	Contains last login information.

Related Information

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

The **newpass** subroutine.

Security

.maildelivery File for MH

Purpose

Specifies actions to be taken when mail is received.

Description

The **$HOME/.maildelivery** file contains a list of actions the **slocal** command performs on received mail. The **slocal** command reads the **$HOME/.maildelivery** file and performs the specified actions when you activate it.

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. The following five 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. A # (pound sign) in the first column indicates a comment. The file is read from beginning to end, 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 **$HOME/.maildelivery** file cannot be found or does not deliver the message, the **/etc/mh/maildelivery** file is used in the same manner. If the message has still not been delivered, it is put in the user's mail drop. The default mail drop is the **/usr/mail/$USER** file.

The MH package contains four standard programs that can be run as receive-mail hooks: the **rcvdist**, **rcvpack**, **rcvstore**, and **rcvtty** commands.

Parameters

`Field` Specifies a header component to be searched for 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. 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, the action is considered successful. Prior to being given to the shell, the string is expanded with the following built-in variables:

$(*Address***)**	Address used to deliver the message.
$(*Size***)**	Size of the message in bytes.
$(*reply-to***)**	Either the `Reply-To:` or `From:` header component of the message.

When a process is started 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 formula for determining 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, it is assumed that the action succeeded. Otherwise, the action is assumed unsuccessful.

qpipe or ^ Acts similarly 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, 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"` Specifies the file to which the message can be appended if you use the **file** value for the *Action* parameter.

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

1. To save a message in a particular file, enter:

   ```
   From george file A george.mail
   ```

   This example 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, enter:

   ```
   addr manager > R proj_X/statlog
   ```

   This example directs the **slocal** command to search the address fields in messages. When it finds a message for 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:

   ```
   * - | R "/usr/lib/mh/rcvtty /home/sarah/allmail"
   ```

   In this example, the `/home/sarah/allmail` file contains the line:

   ```
   echo "You have mail\n"
   ```

   The `/home/sarah/allmail` file must have execute permission. When you have mail, the words `You have mail` are displayed on your console.

4. To forward a copy of a message, enter:

   ```
   addr manager | A "/usr/lib/mh/rcvdist amy"
   ```

   This example 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:

   ```
   default - > ? mailbox
   ```

   This example directs the **slocal** command to find all undelivered messages. The `-` (dash) is a placeholder for the *Pattern* parameter. The `>` (greater than sign) 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 the Base Operating System.

Files

$HOME/.forward	Searched by the **sendmail** command when mail is received, contains either the path of a machine to which to forward mail or a line to start the **slocal** command.
/usr/mail/$USER	Provides the default mail drop.
/usr/lib/mh/slocal	Contains the **slocal** command that reads the **.maildelivery** file.
/etc/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.

mhl.format File

Purpose

Controls the output format of the **mhl** command.

Description

The **/etc/mh/mhl.format** file controls the output format of the **mhl** command when the **mhl** command functions as the message listing program. The **/etc/mh/mhl.format** file is the default attributes file. The **mhl.digest**, **mhl.forward**, and **mhl.reply** files must be specified before use.

Each line of the **mhl.format** file must have one of the following forms:

;*Comment*	Contains the comments specified by the *Comment* field that are ignored.
:*ClearText*	Contains text for output (*ClearText*). A line that contains a : (colon) only produces a blank output line.
Component:[*Variable,...*]	Defines the format of the specified *Component*.
Variable[*Variable,...*]	Applies the value specified by the *Variable* field only to the preceding component if the value follows that component. Lines having other formats define the global environment. The entire **mhl.format** file is parsed before output processing begins. Therefore, if the global setting of a variable 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 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 the *Component* parameter the specified number of columns.
OverflowText	*string*	Outputs the *String* parameter 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 the *Component* parameter in all uppercase characters.
NoUppercase	*flag*	Outputs text of the *Component* parameter 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 the *String* parameter as the name for the specified the *Component* parameter instead of the string *Component*.
NoComponent	*flag*	Does not output the string *Component* for the specified *Component* parameter.
Center	*flag*	Centers the *Component* parameter on line. This variable works for one-line components only.
NoCenter	*flag*	Does not center the *Component* parameter.
LeftAdjust	*flag*	Strips off the leading white space characters from each line of text.
NoLeftAdjust	*flag*	Does not strip off the leading white space 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* parameter contains addresses.
DateField	*flag*	The specified *Component* parameter 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.

Examples

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 the screen should be cleared before each page (`clearscreen`). The overflow text should be flagged with the `***` string, and the overflow indentation should be 5 columns.

Implementation Specifics

This file is part of Message Handler in the Base Operating System.

Files

/etc/mh/mhl.format	Specifies the path of the **mhl.format** file.

Related Information

.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

This table describes the profile entry options for the **.mh_profile** file. Only `Path:` is required. Each profile entry is stored in either the **.mh_profile** file or the *UserMHDirectory*/**context** file.

Profile Entry Options for the .mh_profile File			
Profile Entry	**Description**	**Storage File**	**Default Value**
`Path:`	The path for the*UserMHDirectory* directory. The usual location is **$HOME/Mail**.	**mh_profile**	None
`context:`	The location of the MH context file.	**mh_profile**	*UserMHDirectory* **/context**
`Current-Folder:`	Tracks the current open folder.	**context**	**inbox**
`Previous-Sequence:`	The *Messages* or *Message* sequences parameter given to the program. For each name given, the sequence is set to 0. Each message is added to the sequence. If not present or empty, no sequences are defined.	**mh_profile**	None
`Sequence-Negation:`	The string negating a sequence when prefixed to the name of that sequence. For example, if set to `not`, `not seen` refers to all the messages that are not a member of the sequence `seen`.	**mh_profile**	None

`Unseen-Sequence:`	The sequences defined as messages recently incorporated by the **inc** command. For each name given, the sequence is set to 0. If not present, or empty, no sequences are defined. **Note:** The **show** command removes messages from this sequence after viewing.	**mh_profile**	None
`.mh_sequences:`	The file, in each folder, defining public sequences. To disable the use of public sequences, leave the value of this entry blank.	**mh_profile**	**.mh_sequences**
`atr-`*SequenceFolder*`:`	Tracks the specified sequence in the specified folder.	**context**	None
`Editor:`	The editor to be used by the **comp**, **dist**, **forw**, and **repl** commands.	**mh_profile**	`prompter`
`Msg-Protect:`	Defines octal protection bits for message files. The **chmod** command explains the default values.	**mh_profile**	`0644`
`Folder-Protect:`	Defines protection bits for folder directories. The **chmod** command explains the default values.	**mh_profile**	`0711`
Program`:`	Sets default flags to be used when the MH program specified by the MH program field is started. For example, override the `Editor:` profile entry when replying to messages by entering: `repl: -editor /usr/bin/ed`	**mh_profile**	None
LastEditor`-next:`	The default editor after the editor specified by the `Editor:` field has been used. This takes effect at the `What now?` field of the **comp**, **dist**, **forw**, and **repl** commands. If you enter the **editor** command without a parameter to the `What now?` field, the editor specified by the *LastEditor*`-next:` field is used.	**mh_profile**	None
`Folder-Stack:`	The contents of the folder stack of the **folder** command.	**context**	None

`Alternate-Mailboxes:`	Indicates your address to the **repl** and **scan** commands. The **repl** command is given the addresses to include in the reply. The **scan** command is informed the message originated from you. Host names 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. Enter an * (asterisk) at either end or both ends of the host mailbox to indicate pattern matching. **Note:** Addresses must be separated by a comma.	**mh_profile**	**$LOGNAME**
`Draft-Folder:`	Indicates a default draft folder for the **comp**, **dist**, **forw**, and **repl** commands.	**mh_profile**	None
`digest- issue-`*List*`:`	Indicates to the **forw** command the last issue of the last volume sent for the digest *List*.	**context**	None
`digest-volume-`*List*`:`	Indicates to the **forw** command the last volume sent for the digest *List*.	**context**	None
`MailDrop:`	Indicates to the **inc** command your mail drop, if different from the default. This is superseded by the **$MAILDROP** environment variable.	**mh_profile**	**/usr/mail/$USER**
`Signature:`	Indicates to the **send** command your mail signature. This is superseded by the **$SIGNATURE** environment variable.	**mh_profile**	None

Profile Elements

The following profile elements are used whenever an MH program starts another program. You can use the **.mh_profile** file to select alternate programs.

`fileproc:`	**/usr/bin/refile**
`incproc:`	**/usr/bin/inc**
`installproc:`	**/usr/lib/mh/install-mh**
`lproc:`	**/usr/bin/more**
`mailproc:`	**/usr/bin/mhmail**
`mhlproc:`	**/usr/lib/mh/mhl**
`moreproc:`	**/usr/bin/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/bin/more**
`whatnowproc:`	**/usr/bin/whatnow**
`whomproc:`	**/usr/bin/whom**

Environment Variables

$MH	Specifies a profile for an MH program to read. When you start an MH program, it reads the **.mh_profile** file by default. Use the **$MH** environment variable to specify a different profile. If the file of the **$MH** environment variable does not begin with a / (slash), it is presumed to start in the current directory. The / indicates the file is absolute.
$MHCONTEXT	Specifies a context file that is different from the normal context file specified in the MH profile. If the value of the **$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	Specifies 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 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 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 file system.
$mhdraft	Specifies the path name 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.

Examples

The following example has the mandatory entry for the `Path:`field. The option `-alias aliases` is used when both the **send** and **ali** commands are started. 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 **vi**.

```
Path:          Mail
send:         -alias aliases
ali:          -alias aliases
Msg-Protect: 600
Signature:     Dan Carpenter
Editor:        /usr/bin/vi
```

Implementation Specifics

This file is part of Message Handler in the Base Operating System.

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, **editor** 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,

mibII.my File

Purpose

Provides sample input to the **mosy** command.

Description

The **/usr/samples/snmpd/mibII.my** file is a sample input file to the **mosy** command, which creates an objects definition file for use by the **snmpinfo** command. The **mosy** compiler requires its input file to contain the ASN.1 definitions as described in the Structure and Identification of Management Information (SMI) RFC 1155 and the Management Information Base (MIB) RFC 1213. The **mibII.my** file contains the ASN.1 definitions from the MIB RFC 1213 (MIB II). RFC is the abbreviation for Request for Comments.

Comments are specified by **- -** (two dashes). A comment can begin at any location after the comment sign and extend to the end of the line.

The **mibII.my** file begins with a definition of the SNMP subtree of the MIB, as assigned by the Internet Activities Board (IAB). This definition contains the name of the RFCs from which the ASN.1 definitions are obtained.

```
RFC1213-MIB {iso org(3) dod(6) internet(1) mgmt(2) 1 }

DEFINITIONS ::= BEGIN

IMPORTS
        mgmt, NetworkAddress, IpAddress,
        Counter, Gauge, TimeTicks
        FROM RFC1155-SMI
        OBJECT-TYPE
        from RFC-1213;

mib-2   OBJECT IDENTIFIER ::= { mgmt 1 }-- MIB-II

system           OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces       OBJECT IDENTIFIER ::= { mib-2 2 }
at               OBJECT IDENTIFIER ::= { mib-2 3 }
ip               OBJECT IDENTIFIER ::= { mib-2 4 }
icmp             OBJECT IDENTIFIER ::= { mib-2 5 }
tcp              OBJECT IDENTIFIER ::= { mib-2 6 }
udp              OBJECT IDENTIFIER ::= { mib-2 7 }
egp              OBJECT IDENTIFIER ::= { mib-2 8 }
-- cmot          OBJECT IDENTIFIER ::= { mib-2 9 }
transmission     OBJECT IDENTIFIER ::= { mib-2 10}
snmp             OBJECT IDENTIFIER ::= { mib-2 11}
```

The file must contain the ASN.1 definition for each MIB variable. The ASN.1 definition is presented in an **OBJECT-TYPE** macro.

Following is the format of an **OBJECT-TYPE** macro:

```
ObjectDescriptor          OBJECT-TYPE
        SYNTAX          ObjectSyntax
        ACCESS          AccessMode
        STATUS          StatusType
        DESCRIPTION   Description
        ::= {ObjectGroup Entry}
```

The following definitions describe the pieces of the macro:

ObjectDescriptor Indicates the textual name assigned to the MIB variable being defined. See RFC 1155 for the definition of the *ObjectDescriptor* variable.

ObjectSyntax Indicates the abstract syntax for the object type. It must be one of:

- INTEGER
- OCTET STRING or DisplayString
- OBJECT IDENTIFIER
- NULL
- Network Address
- Counter
- Gauge
- TimeTicks
- Opaque

See RFC 1155 for definitions of each *ObjectSyntax* variable.

AccessMode Specifies the permissions of the object, which can be either:

- read-only
- read-write
- write-only
- not-accessible

See RFC 1155 for definitions of each *AccessMode* variable.

StatusType Specifies the status of the object, which can be either:

- mandatory
- optional
- deprecated
- obsolete

See RFC 1155 for definitions of each *StatusType* variable.

Description Specifies a textual description of the purpose of the MIB variable being defined.

ObjectGroup Defines the object group for this MIB variable. The *ObjectGroup* variable identifies the subtree for the MIB variable. See RFC 1213 for information on object groups.

Entry Defines the unique location of the MIB variable in the *ObjectGroup* variable.

The *ObjectGroup* and *Entry* variables are used to specify the unique numerical object identifier for each MIB variable. See RFC 1155 for an explanation of the object identifier.

See RFC 1155 for further information on the **OBJECT-TYPE** macro.

This sample **mibII.my** file was created by extracting the definitions from Chapter 6, "Definitions," of RFC 1213. This file is shipped as **/usr/samples/snmpd/mibII.my**.

Examples

The following example of an **OBJECT-TYPE** macro describes the `sysDescr` managed object:

```
sysDescr                      OBJECT-TYPE
        SYNTAX                DisplayString (SIZE (0..255))
        ACCESS                read-only
        STATUS                mandatory
        DESCRIPTION           A textual description of the entity.
                              This value should include the full name and
                              version identification of system’s hardware
                              type,software operating-system, and networking
                              software. It is mandatory that this only
                              contain printable ASCII characters.
        ::= { system 1 }
```

Files

/usr/samples/snmpd/mibII.my	Specifies the path of the **mibII.my** file.
/usr/samples/snmpd/smi.my	Defines the ASN.1 definitions by which the SMI is defined in RFC 1155.
/etc/mib.defs	Defines the Management Information Base (MIB) variables the **snmpd** agent should recognize and handle. This file is in the format which the **snmpinfo** command requires.

Implementation Specifics

This file is part of Simple Network Management Protocol Agent Applications in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

The **mosy** command, **snmpinfo** command.

The **smi.my** file.

Management Information Base (MIB) and Terminology Related to Management Information Base (MIB) Variables in *AIX Version 4.3 Communications Programming Concepts*.

RFC 1155, RFC 1213.

Rose, Marshall T. *The Simple Book, An Introduction to Internet Management.* Englewood Cliffs, NJ, Prentice Hall,

mkuser.default File

Purpose

Contains the default attributes for new users.

Description

The **/usr/lib/security/mkuser.default** file contains the default attributes for new users. This file is an ASCII file that contains user stanzas. These stanzas have 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. The end of each attribute pair and stanza is marked by a new-line character.

There are two stanzas, user and admin, that can contain all defined attributes except the **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.

For a list of the possible user attributes, see the **chuser** command.

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.

Examples

A typical user stanza looks like the following:

```
user:
   pgroup = staff
   groups = staff
   shell = /usr/bin/ksh
   home = /home/$USER
   auth1 = SYSTEM
```

Implementation Specifics

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

Files

/usr/lib/security/mkuser.default	Specifies the path to the file.

Related Information

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

mtstailor File for MH

Purpose

Tailors the Message Handler (MH) environment to the local environment.

Description

The entries located in the **/etc/mh/mtstailor** file specify how MH commands work. The following list describes the 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 four Ctrl + A key sequences 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 four Ctrl + A key sequences 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 0 (zero) uses the lockf system call to perform locks. A value of 1 creates lock names by appending `.lock` to the name of the file being locked. The default is 0 (zero).

`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 **/etc/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 the Base Operating System.

Files

/etc/mh/mtstailor	Contains MH command definitions.

Related Information

The **sendmail** command.

mrouted.conf File

Purpose

Default configuration information for the multicast routing daemon **mrouted**.

Description

The **/etc/mrouted.conf** configuration file contains entries that provide configuration information used by **mrouted**. You can specify any combination of these entries in this file.

The file format is free-form; white space and newline characters are not significant. The **phyint**, **tunnel**, and **name** entries can be specified more than once. The **boundary** and **altnet** values can be specified as many times as necessary.

The following entries and their options can be used in the **mrouted.conf** file:

phyint *local_addr* [**disable**] [**metric** *m*] [**threshold** *t*] [**rate_limit** *b*] [**boundary**

(*boundary_name* | *scoped_addr*/*mask_len*)] [**altnet** *network/mask_len*]

The **phyint** entry can be used to disable multicast routing on the physical interface identified by the local IP address *local_addr*, or to associate a non-default metric or threshold with the specified physical interface. The local IP address can be replaced by the interface name (for example, `le0`). If a physical interface is attached to multiple IP subnets, describe each additional subnet with the **altnet** option. **Phyint** entries must precede **tunnel** entries.

The options for the **phyint** entry and the actions they generate are as follows:

local_addr
: Specifies the local address, using either an IP address or an interface name, such as `en0`.

disable
: Disables multicast routing on the physical interface identified by *local_addr*.

metric *m*
: Specifies the "cost" associated with sending a datagram on the given interface or tunnel. This option can be used to influence the choice of routes. The default value of *m* is **1**. Metrics should be kept as small as possible, because **mrouted** cannot route along paths with a sum of metrics greater than 31.

threshold *t*
: Specifies the minimum IP time-to-live (TTL) required for a multicast datagram to be forwarded to the given interface or tunnel. This option controls the scope of multicast datagrams. (The TTL of forwarded packets is compared only to the threshold, it is not decremented by the threshold.) The default value of *t* is **1**. In general, all **mrouted** daemons connected to a particular subnet or tunnel should use the same metric and threshold for that subnet or tunnel.

rate_limit *b*
: Specifies a bandwidth in Kilobits/second, which is allocated to multicast traffic. The default value of *b* is **500** Kbps on tunnels, and **0** (unlimited) on physical interfaces.

boundary *boundary_name*|*scoped_addr/mask_len*
: Configures an interface as an administrative boundary for the specified scoped address. Packets belonging to this address are not forwarded on a scoped interface. The **boundary** option accepts either a boundary name or a scoped address and mask length. The *boundary_name* is the name assigned to a boundary with the **name** entry. The *scoped_addr* value is a multicast address. The *mask_len* value is the length of the network mask.

altnet *network*/*mask_len*
: Specifies an additional subnet (*network*) attached to the physical interface described in the **phyint** entry. *mask_len* is the length of the network mask.

tunnel *local_addr remote_addr* [**metric** *m*] [**threshold** *t*] [**rate_limit** *b*] [**boundary** {*boundary_name* | *scoped_addr*/*mask_len*}] [**altnet** *network*/*mask_len*]

The **tunnel** entry can be used to establish a tunnel link between the local IP address (*local_addr*) and the remote IP address (*remote_addr*), and to associate a non-default metric or threshold with that tunnel. The local IP address can be replaced by the interface name (for example, `le0`). The remote IP address can be replaced by a host name, if and only if the host name has a single IP address associated with it. The tunnel must be set up in the **mrouted.conf** files of both routers before it can be used. The **phyint** entry can be used to disable multicast routing on the physical address interface identified by the local IP address *local_addr* , or to associate a non-default metric or threshold with the specified physical interface. The local IP address can be replaced by the interface name (for example, `le0`). If a physical interface is attached to multiple IP subnets, describe each additional subnet with the **altnet** option. **Phyint** entries must precede **tunnel** entries.

For a description of the options used with the **tunnel** entry, see the preceding option descriptions in the **phyint** entry.

cache_lifetime *ct*
: The **cache_lifetime** entry determines the amount of time that a cached multicast route stays in the kernel before timing out. The value of *ct* is in seconds, and should lie between 300 (five minutes) and 86400 (one day). The default value is **300** seconds **.**

pruning *state*
: The **pruning** entry enables **mrouted** to act as a non-pruning router. The value of *state* can be either **on** or **off** . You should configure your router as a non-pruning router for test purposes only. The default mode is **on** , which enables pruning.

name *boundary_name scoped_addr/mask-len*

The **name** entry lets you assign names to boundaries to make it easier to configure. The **boundary** option on the **phyint** and **tunnel** entries accepts either a boundary name or a scoped address. The *boundary_name* is the name you want to give to the boundary. The *scoped_addr* value is a multicast address. The *mask_len* value is the length of the network mask.

Example

This example shows a configuration for a multicast router at a large school.

```
#
# mrouted.conf
#
# Name our boundaries to make it easier
name LOCAL 239.255.0.0/16 name EE 239.254.0.0/16
#
# le1 is our gateway to compsci, don’t forward our
#   local groups to them
phyint le1 boundary LOCAL
#
# le2 is our interface on the classroom network,
#   it has four different length subnets on it.
# Note that you can use either an IP address or an
#   interface name
phyint 172.16.12.38 boundary EE altnet 172.16.15.0/26
  altnet 172.16.15.128/26 altnet 172.16.48.0/24
#
# atm0 is our ATM interface, which doesn’t properly
# support multicasting
phyint atm0 disable
 #
# This is an internal tunnel to another EE subnet.
# Remove the default tunnel rate limit, since this tunnel
#   is over ethernets
tunnel 192.168.5.4 192.168.55.101 metric 1 threshold 1
  rate_limit 0
# This is our tunnel to the outside world.
tunnel 192.168.5.4 10.11.12.13 metric 1 threshold 32
  boundary LOCAL boundary EE
```

Implementation Specifics

netgroup File for NIS

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. The *universal (, ,)* field 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.

Examples

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 (BOS) Runtime.

Files

/etc/netgroup	Specifies the path of the file.

Related Information

The **makedbm** command.

The **ypserv** daemon.

Mounting an NFS File System Explicitly, Network File System Overview, and Network Information Service Overview in *AIX Version 4.3 System Management Guide: Communications and Networks*.

List of NIS Programming References in *AIX Version 4.3 System Management Guide: Communications and Networks*.

netmasks File for NIS

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 8 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. If this file resides on your system, however, NIS will create a map for it.

Files

/etc/netmasks	Specifies the path of the file.

Related Information

Network File System Overview in *AIX Version 4.3 System Management Guide: Communications and Networks*.

netsvc.conf File

Purpose

Specifies the ordering of certain name resolution services.

Description

The **/etc/netsvc.conf** file is used to specify the ordering of certain services in AIX; specifically, name resolution for **sendmail**, the **gethostbyname**, **gethostaddr**, and **gethostent** subroutines, in addition to alias resolution for **sendmail**.

AIX offers several services for resolving host names and aliases. **gethostbyname**, **gethostbyaddr**, and **gethostent** use the services for resolving names. A default is set to determine the order in which these services are tried for resolving host names and Internet Protocol (IP) addresses.

Resolving Names

The default order can be overwritten by creating the configuration file, **/etc/netsvc.conf** and specifying the desired ordering. To specify the host ordering, enter:

The default and **/etc/irs.conf** order can be overwritten by creating the configuration file, **/etc/netsvc.conf** and specifying the desired ordering. To specify the host ordering, enter:

```
hosts = value [,
value}
```

where *value* can be {`bind`|`local`|`nis`|`bind4`|`bind6`|`local4`|`local6`|`nis4`|`nis6`]

bind	Uses BIND/DNS services for resolving names
local	Searches the local /etc/hosts file for resolving names
nis	Uses NIS services for resolving names
bind4	Uses BIND/DNS services for resolving only IPV4 addresses
bind6	Uses BIND/DNS services for resolving only IPV6 addresses
local4	Searches the local **/etc/hosts** file for resolving only IPV4 addresses
local6	Searches the local **/etc/hosts** file for resolving only IPV6 addresses
nis4	Uses NIS services for resolving only IPV4 addresses
nis6	Uses NIS services for resolving only IPV6 addresses

The environment variable *NSORDER* overrides the host settings in the **/etc/netsvc.conf** file.

Resolving Aliases

The **sendmail** program searches the local file **/etc/aliases** or uses NIS, if specified, for resolving aliases. The default can be overwritten by specifying how to resolve aliases in the **/etc/netsvc.conf** file. To specify alias ordering to **sendmail**, enter:

```
alias = value [,
value}
```

where `value` can be {`files`|`nis`]

files	Searches the local **/etc/aliases** file for the alias
nis	Uses NIS services for resolving alias

The order is specified on one line with values separated by commas. White spaces are permitted around the commas and the equal sign. The values specified and their ordering are dependent on the network configuration.

Examples

To use only **/etc/hosts** for resolving names, enter:

```
hosts = local
```

If you use **/etc/hosts** for resolving names, but the name cannot be found in **/etc/hosts**, then NIS can be used (NIS should be running if you specify it.) Enter:

```
hosts = local , nis
```

To use NIS for resolving names, make it authoritative, and use BIND, enter:

```
hosts = nis = auth , bind
```

To override default and use only NIS for resolving aliases in **sendmail**, enter:

```
aliases = nis
```

Files

/etc/netsvc.conf	Specifies the path to the file.

Related Information

The **sendmail** command.

The **gethostbyname** subroutine, **gethostbyaddr** subroutine, and **gethostent** subroutine.

networks File for NFS

Purpose

Contains information about networks on the NFS Internet network.

Description

The **/etc/networks** file contains information regarding the known networks that make up the Internet network. 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 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 that search the file.

Implementation Specifics

This file is not supported by AIX. However, if this file resides on your system, Network Information Services (NIS) software will create a map for it.

Files

/etc/networks	Specifies the path of the file.

Related Information

NFS Services in *AIX Version 4.3 System Management Guide: Communications and Networks*.

List of NFS Files.

NLSvec File

Purpose

Encodes PostScript fonts for the ISO8859-1 codeset characters that have code points of more than 127 decimal.

Description

The **/usr/lib/ps/NLSvec** file can contain optional comments, optional code sets, and optional character encodings.

If a line begins with an * (asterisk), it is treated as a comment.

If a specified codeset is used, it must precede all character encodings. If a code set is not specified, the default is ISO8859-1. A specified code set uses the following syntax:

x codeset *CodeSetName*

x	Use a lowercase letter.
codeset	Use all lowercase letters.
CodeSetName	Use any valid code set name available for use with the **iconv** command.

A character encoding uses the following syntax:

CodePoint PostscriptFontPosition PostscriptCharacterName

CodePoint	Displays the decimal code point for the character.
PostScriptFontPosition	Displays the new encoding for that character within the PostScript fonts. The encoding can be octal or decimal.
PostScriptCharacterName	Displays the PostScript character name.

The PostScript assigned character encodings as well as the character names can be found in the following book:

Adobe Systems Incorporated. *PostScript Language Reference Manual, Second Edition*. Reading, MA: Addison-Wesley.

Examples

Notes:

1. Following is an example of a specified codeset:

```
x codeset ISO8859-1
```

2. Following is an example of a character encoding:

```
161 0241 exclamdown
```

International Character Support

By default, the output code set for the TranScript commands is ISO8859-1. The output code set can be specified with the **NLSvec** file. For the **enscript**, **ps4014**, **ps630**, and **psplot** TranScript commands, the input codeset is determined from the current locale. The mapping of characters outside the ASCII range is determined through the **iconv** subroutine using the input and output code sets. If there is no corresponding **iconv** converter, the commands treat the input data as if it were produced in ISO8859-1. This means that ASCII data is output correctly for all locales and codesets. For multibyte locales with no **iconv** converters to ISO8859-1 each byte of a multibyte character is treated as individual characters of the ISO8859-1 form. The only exception to this is the **enscript** command, which translates characters rather then bytes in the current locale through the mapping in the **NLSvec** file.

The following table lists the characters from the IBM-850 code set, which does not map directly to the ISO8859-1 code set through the **iconv** subroutine. The following characters would be mapped to 26 (0x1A) by the **iconv** subroutine and thus be discarded on output. It is possible to define an alternative **NLSvec** file for the IBM-850 code set so that more of the characters can be output on a PostScript device. The characters marked with an * (asterisk) before the character name are normally available in a PostScript font.

Code Point	Character Name
159 (0x9F)	* Florin sign, PostScript name: florin
176 (0xB0)	Quarter hashed
177 (0xB1)	Half hashed
178 (0xB2)	Full hashed
179 (0xB3)	Vertical bar
180 (0xB4)	Right-side middle
185 (0xB9)	Double right-side middle
186 (0xBA)	Double vertical bar
187 (0xBB)	Double upper-right corner bar
188 (0xBC)	Double lower-right corner bar
191 (0xBF)	Upper-right corner box
192 (0xC0)	Lower-left corner box
193 (0xC1)	Bottom-side middle
194 (0xC2)	Top-side middle

195 (0xC3)	Left-side middle
196 (0xC4)	Center box bar
197 (0xC5)	Intersection
200 (0xC8)	Double lower-left corner bar
201 (0xC9)	Double upper-left corner bar
202 (0xCA)	Double bottom-side middle
203 (0xCB)	Double top-side middle
204 (0xCC)	Double left-side middle
205 (0xCD)	Double center box bar
206 (0xCE)	Double intersection
213 (0xD5)	* Small dotless i, PostScript name: dotless i
217 (0xD9)	Lower-right corner box
218 (0xDA)	Upper-left corner box
219 (0xDB)	Bright character cell
220 (0xDC)	Bright character cell lower half
223 (0xDF)	Bright character cell upper half
242 (0xF2)	Double underscore
254 (0xFE)	Vertical solid rectangle

Files

XPSLIBDIRX	Specifies the **/usr/lib/ps** directory.
/usr/lib/ps/NLSvec	Contains Adobe TranScript character encodings for the ISO8859-1 code set. This file is the default.
PSVECFILE	Used as an environment variable to define an **NLSvec** file other than the default file.

Implementation Specifics

This file is part of Formatting Tools in the Text Formatting System.

Related Information

ntp.conf File

Purpose

Controls how the Network Time Protocol (NTP) daemon **xntpd** operates and behaves. This file is available only in AIX Version 4.2 or later.

Description

The **ntp.conf** file is a basic configuration file controlling the **xntpd** daemon.

The following options are discussed in this article:

- Configuration Options
- Configuration Authentication Options
- Configuration Access Control Options
- Configuration Monitoring Options
- Miscellaneous Configuration Options

Configuration Options

In the **ntp.conf** file, comments begin with a # character and extend to the end of the line. Blank lines are ignored. Options consist of an initial keyword followed by a list of arguments, which may be optional, separated by whitespace. These options may not be continued over multiple lines. Arguments may be host names, host addresses written in numeric (dotted decimal) form, integers, floating point numbers (when specifying times in seconds) and text strings.

peer [*HostAddress*] [**key** *Number*] [**version** *Number*] [**prefer**]

Specifies that the local server operate in symmetric active mode with the remote server specified by *HostAddress*. In this mode, the local server can be synchronized to the remote server, or the remote server can be synchronized to the local server. Use this method in a network of servers where, depending on various failure scenarios, either the local or remote server host may be the better source of time.

The **key** *Number* specifies that all packets sent to *HostAddress* include authentication fields encrypted using the specified key number. The value of *KeyNumber* is the range of an unsigned 32 bit integer.

The **version** *Number* specifies the version number to use for outgoing NTP packets. The values for *Version* can be **1** or **2**. The default is NTP version 3 implementation.

The **prefer** option marks the host as a preferred host. This host is not subject to preliminary filtering.

server [*HostAddress*] [**key** *Number*] [**version** *Number*] [**prefer**] [**mode** *Number*]

Specifies that the local server operate in client mode with the remote server specified by *HostAddress*. In this mode, the local server can be synchronized to the remote server, but the remote server can never be synchronized to the local server.

The **key** *Number* specifies that all packets sent to *HostAddress* include authentication fields encrypted using the specified key number. The value of *KeyNumber* is the range of an unsigned 32 bit integer.

The **version** *Number* specifies the version number to use for outgoing NTP packets. The values for *Version* can be **1** or **2**. The default is NTP version 3 implementation.

The **prefer** argument marks the host as a preferred host. This host is not subject to preliminary filtering.

broadcast [*HostAddress*] [**key** *Number*] [**version** *Number*] [**ttl** *Number*]

	Specifies that the local server operate in broadcast mode where the local server sends periodic broadcast messages to a client population at the broadcast/multicast address specified by *HostAddress*. Ordinarily, this specification applies only to the local server operating as a transmitter. In this mode, *HostAddress* is usually the broadcast address on [one of] the local network[s] or a multicast address. The address assigned to NTP is 224.0.1.1; presently, this is the only number that should be used. The **key** *Number* specifies that all packets sent to *HostAddress* include authentication fields encrypted using the specified key number. The value of *Number* is the range of an unsigned 32 bit integer. The **version** *Number* specifies the version number to use for outgoing NTP packets. The values for *Version* can be **1** or **2**. The default is NTP version 3 implementation. The **ttl** *Number* is used only with the broadcast mode. It specifies the time-to-live (TTL) to use on multicast packets. This value defaults to 127.
broadcastclient	Specifies that the local server listen for broadcast messages on the local network in order to discover other servers on the same subnet. When the local server hears a broadcast message for the first time, it measures the nominal network delay using a brief client/server exchange with the remote server, then enters the **broadcastclient** mode, where it listens for and synchronizes to succeeding broadcast messages.
multicastclient [*IPAddress* ...]	Works like **broadcastclient** configuration option, but operates using IP multicasting. If you give one or more IP addresses, the server joins the respective multicast group(s). If you do not give an IP address, the IP address assumed is the one assigned to NTP (224.0.1.1).

driftfile *Filename*

Specifies the name of the file used to record the frequency offset of the local clock oscillator. The **xntpd** daemon reads this file at startup, if it exists, in order to set the initial frequency offset and then updates it once per hour with the current offset computed by the daemon. If the file does not exist or you do not give this option, the initial frequency offset assumed is zero. In this case, it may take some hours for the frequency to stabilize and the residual timing errors to subside. The file contains a single floating point value equal to the offset in parts-per-million (ppm).

Note: The update of the file occurs by first writing the current drift value into a temporary file and then using **rename???** to replace the old version. The **xntpd** daemon must have write permission in the directory of the drift file, and you should avoid file system links, symbolic or otherwise.

enable auth | **bclient** | **pll** | **monitor** | **stats** [...]

Enables various server options. Does not affect arguments not mentioned.

The **auth** option causes the server to synchronize with unconfigured peers only if the peer has been correctly authenticated using a trusted key and key identifier. The default for this argument is disable (off).

The **bclient** option causes the server to listen for a message from a broadcast or multicast server, following which an association is automatically instantiated for that server. The default for this argument is disable (off).

The **pll** option enables the server to adjust its local clock, with default enable (on). If not set, the local clock free-runs at its intrinsic time and frequency offset. This option is useful when the local clock is controlled by some other device or protocol and NTP is used only to provide synchronization to other clients.

The **monitor** option enables the monitoring facility, with default enable (on).

The **stats** option enables statistics facility filegen, with default enable (on).

disable auth | **bclient** | **pll** | **monitor** | **stats** [...]

Disables various server options. Does not affect arguments not mentioned. The options are described under the **enable** subcommand.

Configuration Authentication Options

keys *Filename*	Specifies the name of a file which contains the encryption keys and key identifiers used by the **xntpd** daemon when operating in authenticated mode.
trustedkey *Number* [*Number* ...]	Specifies the encryption key identifiers which are trusted for the purposes of authenticating peers suitable for synchronization. The authentication procedures require that both the local and remote servers share the same key and key identifier for this purpose, although you can use different keys with different servers. Each *Number* is a 32 bit unsigned integer. **Note:** The NTP key 0 is fixed and globally known. To perform meaningful authentication, the 0 key should not be trusted.
requestkey *Number*	Specifies the key identifier to use with the **xntpdc** query/control program that diagnoses and repairs problems that affect the operation of the **xntpd** daemon. The operation of the **xntpdc** query/control program is specific to this particular implementation of the **xntpd** daemon and can be expected to work only with this and previous versions of the daemon. Requests from a remote **xntpdc** program which affect the state of the local server must be authenticated, which requires both the remote program and local server share a common key and key identifier. The value of *Number* is a 32 bit unsigned integer. If you do not include **requestkey** in the configuration file, or if the keys do not match, such requests are ignored.
controlkey *Number*	Specifies the key identifier to use with the **ntpq** query program, that diagnoses problems that affect the operation of the **xntpd** daemon. The operation of the **ntpq** query program and the **xntpd** daemon conform to those specified in RFC 1305. Requests from a remote **ntpq** program which affect the state of the local server must be authenticated, which requires both the remote program and local server share a common key and key identifier. The value of *Number* is a 32 bit unsigned integer. If you do not include **controlkey** in the configuration file, or if the keys do not match, such requests are ignored.
authdelay *Seconds*	Specifies the amount of time it takes to encrypt an NTP authentication field on the local computer. This value corrects transmit timestamps when using authentication on outgoing packets. The value usually lies somewhere in the range 0.0001 seconds to 0.003 seconds, though it is very dependent on the CPU speed of the host computer.

Configuration Access Control Options

The **xntpd** daemon inserts default restriction list entries, with the parameters **ignore** and **ntpport**, for each of the local host's interface addresses into the table at startup to prevent the server from attempting to synchronize to its own time. A default entry is also always present, though if it is otherwise unconfigured it does not associate parameters with the default entry (everything besides your own NTP server is unrestricted).

While this facility may be useful for keeping unwanted or broken remote time servers from affecting your own, DO NOT consider it an alternative to the standard NTP authentication facility.

restrict *Address* [**mask** *Number* | **default**] [*Parameter* ...]

Specifies the restrictions to use on the given address. The **xntpd** daemon implements a general purpose address-and-mask based restriction list. The **xntpd** daemon sorts this list by address and by mask, and searches the list in this order for matches, with the last match found defining the restriction flags associated with the incoming packets. The **xntpd** daemon uses the source address of incoming packets for the match, doing a logical and operation with the 32 bit address and the mask associated with the restriction entry. It then compares it with the entry's address (which has also been and'ed with the mask) to look for a match. The **mask** option defaults to 255.255.255.255, meaning that *Address* is treated as the address of an individual host. A default entry (address 0.0.0.0, mask 0.0.0.0) is always included and is always the first entry in the list. The text string **default**, with no mask option, may be used to indicate the default entry.

In the current implementation, *Parameter* always restricts access. An entry with no *Parameter* gives free access to the server. More restrictive *Parameters* will often make less restrictive ones redundant. The *Parameters* generally restrict time service or restrict informational queries and attempts to do run time reconfiguration of the server. You can specify one or more of the following value for *Parameter*:

ignore
Specifies to ignore all packets from hosts which match this entry. Does not respond to queries nor time server polls.

limited
Specifies that these hosts are subject to limitation of number of clients from the same net. Net in this context refers to the IP notion of net (class A, class B, class C, and so on). Only accepts the first **client_limit** hosts that have shown up at the server and that have been active during the last **client_limit_period** seconds. Rejects requests from other clients from the same net. Only takes into account time request packets. Private, control, and broadcast packets are not subject to client limitation and therefore do not contribute to client count. The monitoring capability of the **xntpd** daemon keeps a history of clients. When you use this option, monitoring remains active. The default value for **client_limit** is 3. The default value for **client_limit_period** is 3600 seconds.

lowpriotrap
Specifies to declare traps set by matching hosts to low-priority status. The server can maintain a limited number of traps (the current limit is 3), assigned on a first come, first served basis, and denies service to later trap requestors. This parameter modifies the assignment algorithm by allowing later requests for normal priority traps to override low-priority traps.

nomodify
Specifies to ignore all NTP mode 6 and 7 packets which attempt to modify the state of the server (run time reconfiguration). Permits queries which return information.

nopeer
Specifies to provide stateless time service to polling hosts, but not to allocate peer memory resources to these hosts.

noquery
Specifies to ignore all NTP mode 6 and 7 packets (information queries and configuration requests) from the source. Does not affect time service.

noserve
Specifies to ignore NTP packets whose mode is not 6 or 7. This denies time service, but permits queries.

notrap
Specifies to decline to provide mode 6 control message trap service to matching hosts. The trap service is a subsystem of the mode 6 control message protocol intended for use by remote event-logging programs.

notrust
Specifies to treat these hosts normally in other respects, but never use them as synchronization sources.

ntpport
Specifies to match the restriction entry only if the source port in the packet is the standard NTP UDP port (123).

clientlimit *Number* — Sets **client_limit**. Specifies the number of clients from the same network allowed to use the server. Allows the configuration of client limitation policy.

clientperiod *Seconds* — Sets **client_limit_period**. Specifies the number of seconds to before considering if a client is inactive and no longer counted for client limit restriction. Allows the configuration of client limitation policy.

Configuration Monitoring Options

File generation sets manage statistical files. The information obtained by enabling statistical recording allows analysis of temporal properties of a server running the **xntpd** daemon. It is usually only useful to primary servers.

statsdir *DirectoryPath* — Specifies the full path of the directory in which to create statistical files. Allows modification of the otherwise constant **filegen** filename prefix for file generation sets used for handling statistical logs.

statistics *Type*... — Enables writing of statistical records. The following are the types of statistics supported:

loopstats — Enables recording of loop filter statistical information. Each update of the local clock outputs a line of the following format to the file generation set named `loopstats`:

```
48773 10847.650 0.0001307 17.3478 2
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next three fields show time offset in seconds, frequency offset in parts-per-million and time constant of the clock-discipline algorithm at each update of the clock.

peerstats — Enables recording of peer statistical information. This includes statistical records of all peers of an NTP server and of the 1-pps signal, where present and configured. Each valid update appends a line of the following format to the current element of a file generation set named `peerstats`:

```
48773 10847.650 127.127.4.1 9714 -0.001605
0.00000 0.00142
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next two fields show the peer address in dotted-quad notation and status, respectively. The status field is encoded in hex in the format described in Appendix A of the NTP specification RFC 1305. The final three fields show the offset, delay and dispersion, all in seconds.

clockstats — Enables recording of clock driver statistical information. Each update received from a clock driver outputs a line of the following form to the file generation set named `clockstats`:

```
49213 525.624 127.127.4.1 93 226
00:08:29.606 D
```

The first two fields show the date (Modified Julian Day) and time (seconds and fraction past UTC midnight). The next field shows the clock address in dotted-quad notation, The final field shows the last timecode received from the clock in decoded ASCII format, where meaningful. You can gather and display a good deal of additional information in some clock drivers.

filegen *Name* [**file** *FileName*] [**type** *TypeName*] [flag flagval] [**link**] [**nolink**] [**enable**] [**disabled**]

Configures setting of generation fileset name. Generation filesets provide a means for handling files that are continuously growing during the lifetime of a server. Server statistics are a typical example for such files. Generation filesets provide access to a set of files used to store the actual data. A file generation set is characterized by its type. At any time, at most one element of the set is being written to. Filenames of set members are built from three elements:

Prefix
: This is a constant filename path. It is not subject to modifications with the **filegen** option. It is defined by the server, usually specified as a compile time constant. You can, however, configure it for individual file generation sets with other commands. For example, you can configure the prefix used with `loopstats` and `peerstats` filegens using the **statsdir** option.

file *FileName*
: The string *FileName* is directly concatenated to the prefix with no intervening slash (/). You can modify this by using the **file** argument to the **filegen** option. To prevent filenames referring to parts outside the filesystem hierarchy denoted by prefix, ".." elements are not allowed in this component

Suffix
: This part reflects individual elements of a fileset. It is generated according to the type of a fileset.

type *TypeName*
: Specifies when and how to direct data to a new element of the set. This way, information stored in elements of a fileset that are currently unused are available for administrational operations without the risk of disturbing the operation of the **xntpd** daemon. Most important, you can remove them to free space for new data produced. The following types are supported:

none
: Specifies that the fileset is actually a single plain file.

pid
: Specifies the use of one element of fileset per server running the **xntpd** daemon. This type does not perform any changes to fileset members during runtime; however, it provides an easy way of separating files belonging to different servers running the **xntpd** daemon. The set member filename is built by appending a dot (.) to concatenated prefix and strings denoted in **file** *Name*, and appending the decimal representation of the process id of the **xntpd** server process.

day
: Specifies the creation of one file generation set element per day. The term day is based on UTC. A day is the period between 00:00 and 24:00 UTC. The fileset member suffix consists of a dot (.) and a day specification in the form YYYYMMDD. where YYYY is a 4 digit year number, MM is a two digit month number, and, DD is a two digit day number. For example, all information written at January 10th, 1992 would end up in a file named `PrefixFileName.19920110`.

week
: Specifies the creation of one file generation set element per week. A week is computed as day-of-year modulo 7. The fileset member suffix consists of a dot (.), a four digit year number, the letter `W`, and a two digit week number. For example, all information written at January, 10th 1992 would end up in a file named `PrefixFileName.1992W1`.

month
: Specifies the creation of one file generation set element per month. The fileset member suffix consists of a dot (.), a four digit year number, and a two digit month number. For example, all information written at January, 1992 would end up in a file named `PrefixFileName.199201`.

year
: Specifies the creation of one file generation set element per year. The fileset member suffix consists of a dot (.) and a four digit year number. For example, all information written at January, 1992 would end up in a file named `PrefixFileName.1992`.

age
: Specifies the creation of one file generation set element every 24 hours of server operation. The fileset member suffix consists of a dot (.), the letter `a`, and an eight digit number. This number is the number of seconds of run-time of the server since the start of the corresponding 24 hour period.

enable
: Enables the writing of information to a file generation set.

disabled	Disables the writing of information to a file generation set.
link	Enables the access of the current element of a file generation set by a fixed name by creating a hard link from the current fileset element to a file without *Suffix*. If a file with this name already exists and the number of links of this file is one, it is renamed by appending a dot (.), the letter C, and the pid of the **xntpd** server process. If the number of links is greater than one, the file is unlinked. This allows access of the current file by a constant name.
nolink	Disables access the current element of a file generation set by a fixed name.

Miscellaneous Configuration Options

precision *Number*
: Specifies the nominal precision of the local clock. The *Number* is an integer approximately equal to the base 2 logarithm of the local timekeeping precision in seconds. Normally, the **xntpd** daemon determines the precision automatically at startup, so use this option when the **xntpd** daemon cannot determine the precision automatically.

broadcastdelay *Seconds*
: Specifies the default delay to use when in broadcast or multicast modes. These modes require a special calibration to determine the network delay between the local and remote servers. Normally, this is done automatically by the initial protocol exchanges between the local and remote servers. In some cases, the calibration procedure may fail due to network or server access controls, for example.

Typically for Ethernet, a number between 0.003 and 0.007 seconds is appropriate. The default is 0.004 seconds.

trap *HostAddress* [**port** *Number*] [**interface** *Addess*]

Configures a trap receiver at the given host address and port number for sending messages with the specified local interface address. If you do not specify the port number, the value defaults to 18447. If you do not specify the interface address, the value defaults to the source address of the local interface.

Note: On a multihomed host, the interface used may vary from time to time with routing changes.

Normally, the trap receiver logs event messages and other information from the server in a log file. While such monitor programs may also request their own trap dynamically, configuring a trap receiver ensures that when the server starts, no messages are lost.

setvar *Variable* [**default**]

Specifies to add an additional system variable. You can use these variables to distribute additional information such as the access policy. If **default** follows a variable of the from *Name*=*Value* , then the variable becomes part of the default system variables, as if you used the **ntpq rv** command. These additional variables serve informational purposes only; they are not related to the protocol variables. The known protocol variables always override any variables defined with **setvar**.

There are three special variables that contain the names of all variables of the same group. The **sys_var_list** holds the names of all system variables, the **peer_var_list** holds the names of all peer variables, and the **clock_var_list** holds the names of the reference clock variables.

logconfig *Key*

Controls the amount of output written to syslog or the logfile. By default all output is turned on. You can prefix all *KeyWord*s with = (equal), + (plus) and **-** (dash). You can control four classes of messages: sys, peer, clock, and sync. Within these classes, you can control four types of messages:

info — Outputs informational messages that control configuration information.

events — Outputs event messages that control logging of events (reachability, synchronization, alarm conditions).

status — Outputs statistical messages that describe mainly the synchronization status.

all — Outputs all messages having to do with the specified class and suppresses all other events and messages of the classes not specified.

You form the *KeyWord* by concatenating the message class with the event class. To just list the synchronization state of **xntp** and the major system events, enter:

```
logconfig =syncstatus +sysevents
```

To list all clock information and synchronization information and have all other events and messages about peers, system events and so on suppressed, enter:

```
logconfig =syncall +clockall
```

Files

/etc/ntp.conf — Specifies the path to the file.

Related Information

The **xntpdc** command, the **xntpd** daemon.

ntp.keys File

Purpose

Contains key identifiers and keys controlling authentication of Network Time Protocol (NTP) transactions. This file is available only in AIX Version 4.2 or later.

Description

The **ntp.keys** file contains key identifiers and keys for encryption and decryption of authentication of NTP transactions.

Authentication Key File Format

The NTP standard specifies an extension allowing verification of the authenticity of received NTP packets, and to provide an indication of authenticity in outgoing packets. The **xntpd** daemon implements this by using the MD5 algorithm to compute a message-digest. The specification allows any one of possibly 4 billion keys, numbered with 32 bit key identifiers, to be used to authenticate an association. The servers involved in an association must agree on the key and key identifier used to authenticate their data, although they must each learn the key and key identifier independently.

The **xntpd** daemon reads its keys from a file specified with the **-k** flag or the **keys** statement in the configuration file. You cannot change key number 0 because the NTP standard fixes it as 64 zero bits.

The **ntp.keys** file uses the same comment conventions as the configuration file, **ntp.conf**. Key entries use the following format:

KeyNumber **M** *Key*

where,

KeyNumber	A positive integer
M	Specifies that *Key* is a 1-to-8 character ASCII string, using the MD5 authentication scheme.
Key	The key itself.

One of the keys may be chosen, by way of the **ntp.conf** configuration file **requestkey** statement, to authenticate run-time configuration requests made using the **xntpdc** command. The **xntpdc** command obtains the key from the terminal as a password, so it is generally appropriate to specify the key in ASCII format.

Files

/etc/ntp.keys Specifies the path to the file.

Related Information

The **xntpdc** command, the **xntpd**

objects File

Purpose

Contains the audit events for 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.

Examples

To define the audit events for the **/etc/security/passwd** file, add a stanza to the **/etc/security/audit/objects** file. For example:

```
/etc/security/passwd:
   r = "S_PASSWD_READ"
   w = "S_PASSWD_WRITE"
```

These attributes generate a `S_PASSWD_READ` audit event each time the **passwd** file is read, and a `S_PASSWD_WRITE` audit event each time the file is opened for writing.

Implementation Specifics

This file is part of 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.

Setting Up Auditing in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

Auditing Overview,

/etc/passwd File

Purpose

Contains basic user attributes.

Description

The **/etc/passwd** file contains basic user attributes. This is an ASCII file that contains an entry for each user. Each entry defines the basic attributes applied to a user. When you use the **mkuser** command to add a user to your system, the command updates the **/etc/passwd** file.

> **Note:** Certain system-defined group and user names are required for proper installation and update of the system software. Use care before replacing this file to ensure that no system-supplied groups or users are removed.

An entry in the **/etc/passwd** file has the following form:

*Name***:***Password***:** *UserID***:***PrincipleGroup***:***Gecos***:** *HomeDirectory***:***Shell*

Attributes in an entry are separated by a : (colon). For this reason, you should not use a : (colon) in any attribute. The attributes are defined as follows:

Name	Specifies the user's login name. The user name must be a unique string of 8 bytes or less. There are a number of restrictions on naming users. See the **mkuser** command for more information.
Password	Contains 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	Specifies the user's unique numeric ID. This ID is used for discretionary access control. The value is a unique decimal integer.
PrincipleGroup	Specifies the user's principal 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 unique decimal integer.
Gecos	Specifies 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. The *Gecos* field cannot contain a colon.
HomeDirectory	Specifies 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	Specifies 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, **/usr/bin/sh,** the system shell, is used. The value is a character string that may contain arguments to pass to the initial program.

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

Changing the User File

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

- **chfn**
- **chsh**
- **chuser**
- **lsuser**
- **mkuser**
- **rmuser**

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

The *Password* attribute is always initialized to an * (asterisk), an invalid password. You can set the password with the **passwd** or **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):/home/smith:/usr/bin/sh
   guest:*:200:0::/home/guest:/usr/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. Each attribute is separated by a : (colon).

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):/home/smith:/usr/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 Base Operating System (BOS) Runtime.

Files

/etc/passwd	Contains basic user attributes.
/usr/lib/security/mkuser.default	Contains default attributes for new users.
/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, **chuser** command, **lsuser mkuser** command, **passwd** command, **pwdadm** command, **pwdck** 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.

/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 character.

Each stanza can have the following attributes:

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

lastupdate Specifies 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/user** file for information on password aging.) The **passwd** and **pwdadm** commands normally set this attribute when a password is changed. The value is a decimal integer that can be converted to a text string using the **ctime** subroutine.

flags Specifies the restrictions applied by the **login**, **passwd**, and **su** commands. The value is a list of comma-separated attributes. The **flags** attribute can be left blank or can be one or more of the following values:

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 or the root user. Normally this flag is set implicitly when the **pwdadm** command changes 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.

NOCHECK None of the system password restrictions defined in the **/etc/security/user** file are enforced for this password.

When the **passwd** or **pwdadm** command updates a password, the command adds values for the **password** 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.

Refer to the "Files" section 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 that user 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	file name
S_PASSWD_WRITE	file name

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

The `password` 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 file is part of Base Operating System (BOS) Runtime.

Files

/etc/security/passwd	Specifies the path to the file.
/etc/passwd	Contains basic user attributes.
/etc/security/user	Contains the extended attributes of users.
/etc/security/login.cfg	Contains configuration information for login and user authentication.

Related Information

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

The **ftpd** daemon, **rlogind** daemon.

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

List of Time Data Manipulation Services,

pcnfsd.conf Configuration File

Purpose

Provides configuration options for the **rpc.pcnfsd** daemon.

Description

The **/etc/pcnfsd.conf** file is an ASCII file written by users to add options to the operation of the **rpc.pcnfsd** daemon, which takes no command-line flags.

When started, the **rpc.pcnfsd** daemon checks for the presence of the **pcnfsd.conf** configuration file and conforms its performance to the specified arguments. The following options can be entered in the **pcnfsd.conf** file:

aixargs -B*CharacterPair*
: Controls the printing of burst pages according to the value of the *CharacterPair* variable, as listed below. The first character applies to the header and the second character to the trailer. Possible values are **n** (never), **a** (always), and **g** (group).

HT	Description
nn	No headers, no trailers
na	No headers, trailer on every file
ng	No header, trailer at the end of the job
an	Header on every file, no trailers
aa	Headers and trailers on every file in the job
ag	Header on every file, trailer after job
gn	Header at beginning of job, no trailer
ga	Header at beginning of job, trailer after every file
gg	Header at beginning of job, trailer at end of job

The header and trailer stanzas in the **/etc/qconfig** file define the default treatment of burst pages.

Note: The **-B** flag works exactly like the **-B** flag in the **enq** command. Unlike the **enq** command, however, the **rpc.pcnfsd** daemon does not allow spaces between the **-B** flag and the *CharacterPair* variable.

getjobnum off	Disables the **rpc.pcnfsd** daemon feature that returns job numbers when print jobs are submitted.
printer *Name AliasFor Command*	Defines a PC-NFS virtual printer, recognized only by **rpc.pcnfsd** daemon clients. Each virtual printer is defined on a separate line in the **pcnfsd.conf** file. The following variables are specified with this option.

Name — Specifies the name of the PC-NFS virtual printer to be defined.

AliasFor — Specifies the name of an existing printer that performs the print job.

Note: To define a PC-NFS virtual printer associated with no existing printer, use a single **-** (minus sign) instead of the *AliasFor* variable.

Command — Specifies the command that is run when a file is printed on the *Name* printer. This command is executed by the Bourne shell, using the **-c** option. For complex operations, replace the *Command* variable with an executable shell script.

The following list of tokens and substitution values can be used in the *Command* variable:

Token	Substitution Value
$FILE	The full path name of the print data file. After the command has executed, the file will be unlinked.
$USER	The user name of the user logged in to the client.
$HOST	The host name of the client system.

spooldir *PathName*	Designates a new parent directory, *PathName*, where the **rpc.pcnfsd** daemon stores the subdirectories it creates for each of its clients. The default parent directory is **/var/spool/pcnfs**.
uidrange	Specifies the valid UID (user number) range that the **rpc.pcnfsd** daemon accepts. The default UID range is 101-4294967295.

wtmp off	Disables the login record-keeping feature of the **rpc.pcnfsd** daemon. By default, the daemon appends to the **/var/adm/wtmp** file a record of user logins.

Examples

1. The following sample **pcnfsd.conf** configuration file demonstrates the effects some options have on the operation of the **rpc.pcnfsd** daemon:

   ```
   printer test - /usr/bin/cp $FILE
    /usr/tmp/$HOST-$USER
   printer sandman san ls -l $FILE
   wtmp off
   ```

   The first line establishes a printer test. Files sent to the `test` printer will be copied into the **/usr/tmp** directory. Requests to the `test` PC-NFS virtual printer to list the queue, check the status, or do similar printer operations, will be rejected because a **-** (minus sign) has been given for the *Alias-For* parameter.

   The second line establishes a PC-NFS virtual printer called `sandman` that lists, in long form, the file specifications for the print data file.

   The third line turns off the **rpc.pcnfsd** daemon feature that records user logins.

2. To set a UID range enter:

   ```
   uidrange 1-100,200-50000
   ```

   This entry means that only numbers from 101-199 and over 50000 are invalid UID numbers.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/etc/pcnfsd.conf	Specifies the path of the configuration file.
/var/spool/pcnfs directory	Contains subdirectories for clients of the **pcnfsd** daemon.
/etc/qconfig	Configures a printer queuing system.
/var/adm/wtmp	Describes formats for user and accounting information.

Related Information

The **enq** command.

The **rpc.pcnfsd** daemon.

Bourne Shell in *AIX Version 4.3 System User's Guide: Operating System and Devices*.

Network File System Overview in *AIX Version 4.3 System Management Guide: Communications and Networks*.

List of NFS Files.

portlog File

Purpose

Contains per-port unsuccessful login attempt information and port locks.

Description

The **/etc/security/portlog** file is an ASCII file that contains stanzas of per port unsuccessful login attempt information and port locks. Each stanza has a name followed by a : (colon) that defines the port name. Attributes are in the form **Attribute**=*Value*. Each attribute ends with a new line character and each stanza ends with an additional new line character.

The attributes in the stanzas are as follows:

locktime	Defines the time the port was locked in seconds since the epoch (zero time, January 1, 1970). This value is a decimal integer string.
unsuccessful_login_times	Lists the times of unsuccessful login attempts in seconds since the epoch. The list contains decimal integer strings separated by commas.

These attributes do not have default values. If a value is not specified, the attribute is ignored.

Security

Access Control: This file grants read access to the root user and members of the security group, and write access only to the root user. Access for other users and groups depends upon the security policy of the operating system.

Examples

A typical record looks like the following example for the **/dev/tty0** port:

```
/dev/tty0:
   locktime = 723848478
   unsuccessful_login_times =
723848430,723848450,723848478
```

Implementation Specifics

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

Files

/etc/security/portlog	Specifies the path to the file.
/etc/security/login.cfg	Contains configuration information for login and user authentication.

Related Information

The **chsec** command, **login** command, **su** command.

The **loginfailed** subroutine, **loginrestrictions** subroutine.

pwdhist File

Purpose

Contains password history information.

Description

The **/etc/security/pwdhist.dir** and **/etc/security/pwdhist.pag** files are database files created and maintained by Database Manager (DBM) subroutines. The files maintain a list of previous user passwords.

The **pwdhist** files store information by user name. User names are the keys of the DBM subroutines. The password list contains multiple pairs of a **lastupdate** value and an encrypted, null-terminated password. This password list is a key's associated content and the **lastupdate** value is a 4-byte, unsigned long. The encrypted password is the size of the **PW_CRYPTLEN** value. Thus, an entry in the database file is of the following format:

```
lastupdatepasswordlastupdatepasswordlastupdatepasswor
d...
```

The password list is in descending chronological order, with the most recent password appearing first in the list.

To retrieve a user's password history, use the **dbm_fetch** subroutine. To delete a user's password history, use the **dbm_delete** subroutine.

Security

Access Control: The files grant read and write access only to the root user.

Examples

If user `sally` has the following previous passwords:

```
password = 6PugcayXL.1Rw ; lastupdate =
737161212

password = r5MZvr69mGeLE ;
lastupdate = 746458629
```

the **dbm_fetch** subroutine returns the following entry for the key `sally`:

```
746458629r5MZvr69mGeLE7371612126PugcayXL.1Rw
```

Implementation Specifics

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

Related Information

The **/etc/security/passwd** file, **/etc/security/user** file.

The **passwd** command.

For lists of DBM and NDBM Subroutines, see List of NDBM and

publickey File for NIS

Purpose

Contains public or secret keys for 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 refer to either 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 **publickey** file should only contain data on the 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 (BOS) Runtime.

Related Information

The **chkey** command, **keylogin** command, **newkey** command.

The **keyserv** daemon, **ypupdated** daemon.

Exporting a File System Using Secure NFS, Mounting a File System Using Secure NFS, Network File System Overview,

qconfig File

Purpose

Configures a printer queuing system.

Description

The **/etc/qconfig** file describes the queues and devices available for use by both 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 **LPDEST** or **PRINTER** environment variable is set, the **enq** command uses this queue when it receives no queue parameter. If **LPDEST** contains a value, that value takes precedence over the **PRINTER** environment variable. Destination command-line options always override both variables.

The name of a queue stanza can be from 1 to 20 characters long. Some of the fields and their possible values that can appear in this file are:

`acctfile`	Identifies the file used to save print accounting information. **FALSE**, the default value, 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 value, fcfs, means first-come-first-served. **sjn** means shortest job next.
`up`	Defines the state of the queue. **TRUE**, the default value, indicates that the queue is running. **FALSE** indicates that it is not running.

Note: lp is a BSD standard reserved queue name and should not be used as a queue name in the **qconfig** file.

The following list shows some of the fields and their possible values that appear in the **qconfig** file for remote queues:

`host`	Indicates the remote host where the remote queue is found.
`s_statfilter`	Specifies the short version filter used to translate remote queue status format. The default option, **/usr/lib/lpd/aixshort**, indicates that the remote print server operates on AIX Version 3, and status information will be represented in short format. Other choices are:

/usr/lib/lpd/bsdshort	BSD remote system
/usr/lib/lpd/aixv2short	RT remote system
/usr/lib/lpd/attshort	AT&T remote system

`l_statfilter`	Specifies the long version filter used to translate remote queue status format. The default option, **/usr/lib/lpd/aixlong**, indicates that the remote print server operates on AIX Version 3, and status information will be represented in long format. Other choices are:

/usr/lib/lpd/bsdlong	BSD remote system
/usr/lib/lpd/aixv2long	RT remote system
/usr/lib/lpd/attlong	AT&T remote system

`rq`	Specifies the remote queue name. In a remote print environment, the client configuration should specify the remote queue name or the server. Using the default remote queue name may cause unpredictable results.

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 from 1 to 20 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 value is TRUE.
`backend`	Specifies the full path name of the backend, optionally followed by the flags and parameters to be passed to it. The path names most commonly used are **/usr/lib/lpd/piobe** for local print and **/usr/lib/lpd/rembak** for remote print.
`feed`	Specifies either 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 value, 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. A value of **never**, the default value, 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. In a remote print environment, the default action is to print a header page and not to print a trailer page.
`trailer`	Specifies whether a trailer page prints after each job or group of jobs. A value of **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. In a remote print environment, the default action is to print a header page and not to print a trailer page.

The **qdaemon** process 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 format when the binary version is missing or older than the ASCII version. The binary version is found in the **/etc/qconfig.bin** file.

Note: The **qconfig** file should not be edited while there are active jobs in any queue. Any time the **qconfig** file is changed, jobs submitted prior to the change will be processed before jobs submitted after the change.

Editing includes both manual editing and use of the **mkque**, **rmque**, **chque**, **mkquedev**, **rmquedev**, or **chquedev** command. It is recommended that all changes to the **qconfig** file be made using these commands. However, if manual editing is desired, first issue the **enq -G** command to bring the queuing system and the **qdaemon** to a halt after all jobs are processed. Then edit the **qconfig** file and restart the **qdaemon** with the new configuration.

Examples

1. The batch queue supplied with the system might contain these stanzas:

   ```
   bsh:
     discipline = fcfs
     device = bshdev
   bshdev:
     backend = /usr/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 = /usr/bin/ksh
   bsh2:
     backend = /usr/bin/ksh
   ```

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

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

Files

/etc/qconfig	Contains the configuration file.
/etc/qconfig.bin	Contains the digested, binary version of the **/etc/qconfig** file.
/dev/null	Provides access to the null device.
/usr/lib/lpd/piobe	Specifies the path of the local printer backend.
/usr/lib/lpd/rembak	Specifies the path of the remote printer backend.
/usr/lib/lpd/digest	Contains the program that converts the **/etc/qconfig** file to binary format.

Related Information

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

Understanding the Interaction between qdaemon and the Backend in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*.

rc.boot File

Purpose

Controls the machine boot process.

Description

> **Attention:** Executing the **rc.boot** script on a system that is already running may cause unpredictable results.

The **/sbin/rc.boot** file is a shell script that is called by the simple shell **init** and the standard **init** command to bring up a system. Depending upon the type of boot device, the **rc.boot** file configures devices and also calls the appropriate applications. Appropriate applications include:

- Booting from disk
- Varying on a root volume group
- Enabling file systems
- Calling the BOS installation programs or diagnostics

The **rc.boot** program is only called by an init process.

Implementation Specifics

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

Files

/etc/inittab	Controls the initialization process.
/usr/lib/boot/ssh	Calls the **rc.boot** file.

Related Information

Accessing a System That Will Not Boot in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

rc.tcpip File for TCP/IP

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 err
   or and event logging)
   start /usr/sbin/syslogd "$src_running"
   ```

2. The following stanza starts the **lpd** daemon:

   ```
   #Start up print daemon
   start /usr/sbin/lpd "$src_running"
   ```

3. The following stanza starts the **routed** daemon, but not the **gated** daemon:

```
#Start up routing daemon (only s
tart ONE)
start /usr/sbin/routed "$src_running" -g
#start /usr/sbin/gated "$src_running"
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

The **startsrc** command, **stopsrc** command.

The **gated** daemon, **inetd** daemon, **lpd** daemon, **named** daemon, **portmap** daemon, **routed** daemon, **rwhod** daemon, **sendmail** daemon, **syslogd** daemon, **timed** daemon.

Naming in *AIX Version 4.3 System Management Guide: Communications and Networks*.

Installation and Configuration for TCP/IP in *AIX Version 4.3 System Management Guide: Communications and Networks*.

remote.unknown File for BNU

Purpose

Logs access attempts by unknown remote systems.

Description

The **/usr/sbin/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 **/etc/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 **/var/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/usr/sbin/uucp/remote.unknown	Contains the **remote.unknown** shell script.
/etc/sbin/Permissions	Describes access permissions for remote systems.
/var/spool/uucp/.Admin/Foreign	Lists access attempts by unknown systems.

Related Information

roles File

Purpose

Contains the list of valid roles. This system file only applies to AIX Versions 4.2.1 and later.

Description

The **/etc/security/roles** file contains the list of valid roles. This is an ASCII file that contains a stanza for each system role. Each stanza is identified by a role name followed by a : (colon) and contains attributes in the form *Attribute=Value*. Each attribute pair ends with a newline character as does each stanza.

The file supports a default stanza. If an attribute is not defined, the default value for the attribute is used.

A stanza contains the following attributes:

rolelist	Contains a list of roles implied by this role and allows a role to function as a super-role. If the **rolelist** attribute contains the value of "*role1,role2*", assigning the `role` to a user also assigns the roles of *role1* and *role2* to that user.
authorizations	Contains the list of additional authorizations acquired by the user for this specific role.
groups	Contains the list of groups that a user should belong to in order to effectively use this role. The user must be added to each group in this list for this role to be effective.
screens	Contains a list of SMIT screen identifiers that allow a role to be mapped to various SMIT screens. The default value for this attribute is * (all screens).
msgcat	Contains the file name of the message catalog that contains the one-line descriptions of system roles.
msgnum	Contains the message ID that retrieves this role description from the message catalog.

For a typical stanza, see the "Examples" stanza.

Changing the roles File

You should access this file through the commands and subroutines defined for this purpose. You can use the following commands to change the **roles** file:

- **chrole**
- **lsrole**
- **mkrole**
- **rmrole**

The **mkrole** command creates an entry for each new role in the **/etc/security/roles** file. To change the attribute values, use the **chrole** command. To display the attributes and their values, use the **lsrole** command. To remove a role, use the **rmrole** command.

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

Security

Access Control: This file grants read and write access to the root user, and read access to members of the security group.

Examples

A typical stanza looks like the following example for the `ManageAllUsers` role:

```
ManageAllUsers:

   rolelist = ManageBasicUsers
   authorizations = UserAdmin,RoleAdmin,PasswdAdmin,GroupAdmin
   groups = security
   screens = mkuser,rmuser,!tcpip
```

Implementation Specifics

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

Files

/etc/security/roles	Contains the list of valid roles.
/etc/security/user.roles	Contains the list of roles for each user.
/etc/security/smitacl.group	Contains the group ACL definitions.
/etc/security/smitacl.user	Contains the user ACL definitions.

Related Information

The **chrole** command, **lsrole** command, **mkrole** command, **rmrole** command.

rpc File for NFS

Purpose

Contains the database for Remote Procedure Calls (RPC) program numbers using NFS.

Description

The **/etc/rpc** file contains names that are used in place of RPC program numbers. These names can be read by users. Each line of the file contains the following entries:

Name of Server for the RPC Program	Specifies the name of the server daemon that provides the RPC program.
RPC Program Number	Specifies the number assigned to the program by the RPC protocol.
Aliases	Specifies alternate names by which the service can be requested.

The three entries for each line are entered in the order listed here. Entries can be separated by any number of blanks or tab characters, provided the line does not wrap. Commented lines in the file must begin with a # (pound sign). Characters in a commented line are not interpreted by routines that search the file.

Examples

A sample **/etc/rpc** file follows:

```
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 (BOS) Runtime.

Related Information

File Systems Overview in *AIX Version 4.3 System Management Concepts: Operating System and Devices*.

sendmail.cf File

Purpose

Contains the configuration information for the **sendmail** command.

Description

The **/etc/sendmail.cf** configuration file contains the configuration information for the **sendmail** command. Information contained in this file includes such items as the host name and domain, and the **sendmail** rule sets.

The **/etc/sendmail.cf** file:

- Stores information about the type of mailer programs running.
- Defines how the **sendmail** command rewrites addresses in messages.
- Defines how the **sendmail** command operates in the following environments:
  - Local mail delivery
  - Local area network delivery using TCP/IP
  - Remote delivery using Basic Utilities Network (BNU).

If your environment includes only these types of mail delivery, you can use the supplied **/etc/sendmail.cf** file with few, if any, changes.

Control Lines

The **/etc/sendmail.cf** file consists of a series of control lines, each of which begins with a single character defining how the rest of the line is used. Lines beginning with a space or a tab are continuation lines. Blank lines and lines beginning with a # (pound sign) are comments. Control lines are used for defining:

- Macros and classes for use within the configuration file
- Message headings
- Mailers
- Options for the **sendmail** command

Each of these control line types are discussed in detail below.

Rewrite Rules

The **sendmail** command receives addresses in a number of different formats because different mailers use different formats to deliver mail messages. The **sendmail** command changes the addresses to the format needed to route the message for the mailer program being used. To perform this translation, the **sendmail** command uses a set of rewrite rules, or *rule sets*, that are defined in the **/etc/sendmail.cf** configuration file. Rewrite rules have the following format:

```
Snumber
Rbefore          after
```

where `number` is a integer greater than or equal to zero indicating which ruleset this is, and `before` and `after` are symbolic expressions representing a particular pattern of characters. The line beginning with `R` means "Rewrite the expression `before` so that it has the same format as the expression `after`." **Sendmail** scans through the set of rewrite rules looking for a match on the left-hand side (LHS) of the rule. When a rule matches, the address is replaced by the right-hand side (RHS) of the rule.

> **Note:** There must be at least one TAB character (ASCII code 0x09) between the `before` and `after` sections of the **/etc/sendmail.cf** file. For this reason, any editor that translates TAB characters into a series of spaces (ASCII code 0x20) may not be used to edit the **/etc/sendmail.cf** file. For example, the GNU eMacs editor can corrupt the **sendmail.cf** file, but the vi editor does not.

The **/etc/sendmail.cf** file installed with the **sendmail** command contains enough rules to perform the translation for BNU and TCP/IP networks using a domain address structure. You should not have to change these rules unless connecting to a system that uses a different addressing scheme.

Macro expansions of the form $x are performed when the configuration file is read. Expansions of the form $&x are performed at run time, using a somewhat less general algorithm. This form is intended only for referencing internally defined macros such as $h that are changed at runtime.

Left-Hand Side (LHS) of Rewrite Rules

The left-hand side of rewrite rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

$*	Match zero or more tokens
$+	Match one or more tokens
$-	Match exactly one token
$=x	Match any phrase in class x
$~x	Match any word not in class x

If any of these match, they are assigned to the symbol $n for replacement on the right-hand side, where n is the index in the LHS. For example, if the LHS:

```
$-:$+
```

is applied to the input:

```
UCBARPA:linda
```

the rule will match, and the values passed to the RHS will be:

```
$1  UCBARPA
$2  linda
```

Right-Hand Side (RHS) of Rewrite Rules

When the left-hand side of a rewrite rule matches, the input is deleted and replaced by the right-hand side. Tokens are copied directly from the RHS unless they begin with a dollar sign. Metasymbols are:

$n	Substitute indefinite token **n** from LHS
$[**name**$]	Canonicalize name
$(**map key **$@arguments **$:**default **$)**	Generalized keyed mapping function
$>n	"Call" ruleset **n**
$#mailer	Resolve to mailer
$@host	Specify host
$:user	Specify user

The $n syntax substitutes the corresponding value from a **$+**, **$-**, **$***, **$=**, or **$~** match on the LHS. It may be used anywhere.

A host name enclosed between $[and $] is looked up in the host database(s) and replaced by the canonical name. For example, $[merlin]" might become "merlin.magician" and "$[[128.32.130.2]$]" would become "king.arthur."

The $(... $) syntax is a more general form of lookup; it uses a named map instead of an implicit map. If no lookup is found, the indicated default is inserted; if no default is specified and no lookup matches, the value is left unchanged. The arguments are passed to the map for possible use.

The $>n syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset n. The final value of ruleset n then becomes the substitution for this rule. The $> syntax can only be used at the beginning of the right hand side; it can be only be preceded by $@ or $:.

The $# syntax should only be used in ruleset zero or a subroutine of ruleset zero. It causes evaluation of the ruleset to terminate immediately, and signals to **sendmail** that the address has completely resolved. The complete syntax is:

```
$#mailer $@host $:user
```

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local, the host part may be omitted. The mailer must be a single word, but the host and user may be multi-part. If the mailer is the builtin IPC mailer, the host may be a colon-separated list of hosts that are searched in order for the first working address, exactly like MX (machine exchange) records. The user is later rewritten by the mailer-specific envelope rewrite set and assigned to the $u macro. As a special case, if the value to $# is "local" and the first character of the $: value is "@", the "@" is stripped off, and a flag is set in the address descriptor that causes **sendmail** to not do ruleset 5 processing.

Normally, a rule that matches is retried, that is, the rule loops until it fails. An RHS may also be preceded by a $@ or a $: to change this behavior. A $@ prefix causes the ruleset to return with the remainder of the RHS as the value. A $: prefix causes the rule to terminate immediately, but the ruleset to continue; this can be used to avoid continued application of a rule. The prefix is stripped before continuing.

The $@ and $: prefixes may precede a $> spec. For example:

```
R$+  $: $>7 $1
```

matches anything, passes that to ruleset seven, and continues; the $: is necessary to avoid an infinite loop.

Substitution occurs in the order described; that is, parameters from the LHS are substituted, hostnames are canonicalized, "subroutines" are called, and finally $#, $@, and $: are processed.

Semantics of Rewrite Rule Sets

There are five rewrite sets that have specific semantics.

Ruleset three should turn the address into "canonical form." This form should have the basic syntax:

```
local-part@host-domain-spec
```

Ruleset three is applied by **sendmail** before doing anything with any address.

If no "@" sign is specified, then the host-domain-spec may be appended (box "D" in "Rewrite Set Semantics") from the sender address (if the C flag is set in the mailer definition corresponding to the sending mailer).

Ruleset zero is applied after ruleset three to addresses that are going to actually specify recipients. It must resolve to a {mailer, host, user} triple. The mailer must be defined in the mailer definitions from the configuration file. The host is defined into the $h macro for use in the argv expansion of the specified mailer.

IPC Mailers

Some special processing occurs if the ruleset zero resolves to an IPC mailer (that is, a mailer that has "[IPC]" listed as the Path in the M configuration line. The host name passed after "$@" has MX expansion performed; this looks the name up in DNS to find alternate delivery sites.

The host name can also be provided as a dotted quad in square brackets; for example:

```
[128.32.149.78]
```

This causes direct conversion of the numeric value to a TCP/IP host address.

The host name passed in after the "$@" may also be a colon-separated list of hosts. Each is separately MX expanded and the results are concatenated to make (essentially) one long MX list. The intent here is to create "fake" MX records that are not published in DNS for private internal networks.

As a final special case, the host name can be passed in as a text string in square brackets:

```
[any.internet.addr]
```

This form avoids the MX mapping if the F=0 flag is set for the selected delivery agent.

Note: This is intended only for situations where you have a network *firewall* (a system or machine that controls the access between outside networks and private networks) or other host that will do special processing for all your mail, so that your MX record points to a gateway machine. This machine could then do direct delivery to machines within your local domain. Use of this feature directly violates RFC 1123 section 5.3.5: it should not be used lightly.

Macros in the sendmail.cf File

Macros in the **/etc/sendmail.cf** file are interpreted by the **sendmail** command. A macro is a symbol that represents a value or string. A macro is defined by a `D` command in the **/etc/sendmail.cf** file.

D -- Define Macro

Macros are named with a single character or with a word in {braces}. Single-character names may be selected from the entire ASCII set, but user-defined macros should be selected from the set of uppercase letters only. Lowercase letters and special symbols are used internally. Long names beginning with a lowercase letter or a punctuation character are reserved for use by **sendmail**, so user-defined long macro names should begin with an uppercase letter.

The syntax for macro definitions is:

`Dxval`

where `x` is the name of the macro (which may be a single character or a word in braces) and `val` is the value it should have. There should be no spaces given that do not actually belong in the macro value.

Macros are interpolated using the construct $x, where x is the name of the macro to be interpolated. This interpolation is done when the configuration file is read, except in M lines. The special construct $&x can be used in R lines to get deferred interpolation.

Conditionals can be specified using the syntax:

```
$?x text1 $| text2 $.
```

This interpolates text1 if the macro $x is set, and text2 otherwise. The "else" ($|) clause may be omitted.

Lowercase macro names are reserved to have special semantics, used to pass information in or out of **sendmail**, and special characters are reserved to provide conditionals, and so on. Uppercase names (that is, $A through $Z) are specifically reserved for configuration file authors.

The following macros are defined and/or used internally by **sendmail** for interpolation into argv's for mailers or for other contexts. The ones marked - are information passed into **sendmail**, the ones marked = are information passed both in and out of **sendmail**, and the unmarked macros are passed out of **sendmail** but are not otherwise used internally. AIX Version 4.2 specifically defines the following macros:

$_	RFC1413-validation & IP source route (V8.1 and above).
$a	The origin date in RFC822 format.

$b	The current date in RFC822 format.
$(bodytype)	The ESMTP BODY parameter.
$B	The BITMAP relay.
$c	The hop count.
$(client_addr)	The connecting host's IP address.
$(client_name)	The connecting host's canonical name.
$(client_port)	The connecting host's port name.
$C	The hostname of the DECnet relay (m4 technique).
$d	The current date in UNIX (*ctime*)(3) format.
$e	SMTP greeting message (V8.8 and earlier).
$(envid)	The original DSN envelope ID (V8.8 and above).
$E	X400 relay (unused) (m4 technique).
$f	The sender's address.
$F	FAX relay (m4 technique).
$g	The sender's address relative to the recipient.
$h	Host part of the recipient address.
$H	The mail hub (m4 technique).
$i	The queue identifier.
$j=	The official canonical name.
$k	The UUCP node name (V8.1 and above).
$l	Unix From format (V8.6 and earlier).
$L	Local user relay (m4 technique).
$m	The DNS domain name (V8.1 and above).
$M	Who we are masquerading as (m4 technique).
$n	The error messages sender.
$o	Token separation characters (V8.6 and earlier).
$opMode	The startup operating mode (V8.7 and above).
$p	The *sendmail* process id.
$q-	Default form of the sender address (V8.6 and earlier).
$r	The protocol used.

$R	The relay for unqualified names (m4 technique).
$s	The sender's host name.
$S	The Smart host (m4 technique).
$t	Current time in seconds.
$u	The recipient's username.
$U	The UUCP name to override $k.
$v	The *sendmail* program's version.
$V	The UUCP relay (for class $=V) (m4 technique).
$w	The short name of this host.
$W	The UUCP relay (for class $=W) (m4 technique).
$x	The full name of the sender.
$X	The UUCP relay (for class $=X) (m4 technique).
$y	The home directory of the recipient.
$z	The name of the controlling TTY.
$Y	The UUCP relay for unclassified hosts.
$z	The recipient's home directory.
$Z	The version of this m4 configuration (m4 technique).

There are three types of dates that can be used. The **$a** and **$b** macros are in RFC 822 format; **$a** is the time as extracted from the "Date:" line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no "Date:" line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$b** macro in UNIX (ctime) format. The **$t** macro is the current time in seconds.

The macros **$w**, **$j**, and **$m** are set to the identity of this host. **Sendmail** tries to find the fully qualified name of the host if at all possible; it does this by calling gethostname(2) to get the current hostname and then passing that to gethostbyname(3) which is supposed to return the canonical version of that host name. Assuming this is successful, **$j** is set to the fully qualified name, and **$m** is set to the domain part of the name (everything after the first dot). The **$w** macro is set to the first word (everything before the first dot) if you have a level 5 or higher configuration file; otherwise, it is set to the same value as **$j**. If the canonification is not successful, it is imperative that the config file set **$j** to the fully qualified domain name.

The **$f** macro is the id of the sender as originally determined; when mailing to a specific host, the **$g** macro is set to the address of the sender relative to the recipient. For example, if a user sends to `king@castle.com` from the machine `vangogh.painter.com`, the **$f** macro will be `vincent` and the **$g** macro will be `vincent@vangogh.painter.com`.

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as flag to **sendmail**. It can be defined in the NAME environment variable. The third choice is the value of the "Full-Name:" line in the header if it exists, and the fourth choice is the comment field of a "From:" line. If all of these fail, and if the message is being originated locally, the full name is looked up in the **/etc/passwd** file.

When sending, the **$h**, **$u**, and **$z** macros get set to the host, user, and home directory (if local) of the recipient. The first two are set from the **$@** and **$:** part of the rewrite rules, respectively.

The **$p** and **$t** macros are used to create unique strings (for example, for the "Message-Id:" field). The **$i** macro is set to the queue id on this host; if put into the timestamp line, it can be useful for tracking messages. The **$v** macro is set to be the version number of **sendmail**; this is normally put in timestamps and has been proven useful for debugging.

The **$c** field is set to the "hop count," that is, the number of times this message has been processed. This can be determined by the **-h** flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with **sendmail** and the sending hostname.They can be set together using the **-p** command line flag or separately using the **-M** or **-oM** flags.

The **$_** is set to a validated sender host name. If the sender is running an RFC 1413 compliant IDENT server and the receiver has the IDENT protocol turned on, it will include the user name on that host.

The **$(client_name)**, **$(client_addr)**, and **$(client_port)** macros are set to the name, address, and port number of the connecting host who is invoking **sendmail** as a server. These can be used in the **check_*** rulesets (using the **$&** deferred evaluation form).

Changing the Domain Name Macro

Note: This function is available in AIX Version 3.2.5 and AIX Version 4.1 only.

The domain name macro, **DD**, specifies the full domain name of your local group of hosts. The format of the domain name macro is DD followed by, at most, four period-separated names, for example:

`DDname1.name2.name3.name4`

This macro can be set automatically through the **hostname** command. The **sendmail** command reads what has been set with the **hostname** command and uses it to initialize the host and domain macros and classes. The configuration file macros only need to be changed if you want the **sendmail** host and domain names to be different from those set by the **hostname** command.

To change the domain name macro:

1. Enter the command:

   ```
   vi /etc/sendmail.cf
   ```

2. Find the line beginning with `DD`.

3. Replace what follows `DD` with your domain name. For example, if your domain name is `newyork.abc.com`, enter:

   ```
   DDnewyork.abc.com
   ```

4. Save the file and exit the editor.

Changing the Host Name Macro

The host name macro, **Dw**, specifies the name of your host system used in the return address of all messages you generate. The format of the host name macro is Dw followed by the hostname of this machine, for example:

```
Dwhostname
```

By default, the **sendmail** command reads what has been set with the **hostname** command and uses it to initialize the host and domain name macros and classes. Change the configuration file macros only if you want the **sendmail** command host and domain names to be different from those set by the **hostname** command.

To change the host name macro:

1. Enter the command:

   ```
   vi /etc/sendmail.cf
   ```

2. Find the line beginning with `Dw`.
3. Replace what follows `Dw` with your hostname. For example, if your hostname is `brown`, enter:

   ```
   Dwbrown
   ```

4. Save the file and exit the editor.

> **Note:** If the **Dw** macro is defined, you must also define the **CW** (hostname) class.

Configuration File Revision-Level Macro (DZNumber)

The revision-level **Z** macro tracks changes you make to the **/etc/sendmail.cf** file. When you make a change to the **/etc/sendmail.cf** file, change the value of this macro to show your new version level. You can use any format you wish for the number. For example, if you want the revision-level to be 3.1, make the following entry in the **/etc/sendmail.cf** file:

```
DZ3.1
```

or, use a text string for this macro:

```
DZthree_one
```

Recommended Customizations of the sendmail.cf File

The following is a list of the control lines you may want to configure:

Note: These control lines are available for customization in AIX Version 3.2.5 and AIX Version 4.1 only.

- *HostName* class (**Cw**) and *HostName* macro (**Dw**). By default, the **sendmail** command reads the value returned by the **hostname** command to initialize the system's mail name. Change this class and macro if you want the mail name of the system to be different from the name set by the **hostname** command.
- *DomainName* macro (**DD**) and *DomainName* class (**Cd**).
- Operational logging level (**OL** option).
- Default delivery mode (**Od** option).
- Alias file path (**OA** option).
- Statistics file path (**OS** option).
- Queue directory path (**OQ** option).
- Maximum message retention time in queue (**OT** option).
- Queueing uses of expensive mailers (**Oc** option).
- Configuration file revision level (**Z** macro).

Modifying the sendmail.cf File

Before you modify the **/etc/sendmail.cf** file, make a backup copy. Do this by executing the following command:

```
cp /etc/sendmail.cf /etc/sendmail.cf.working
```

If the changes you make cause the mail system not to work properly, you can return to using a copy of the **/etc/sendmail.cf** file that you know works.

You can modify the **/etc/sendmail.cf** file by using your favorite text editor. However, some editors store tabs as the number of spaces they represent, not the tab character itself. This can cause unexpected results if the tab character is defined as the field-separator character in rule sets. Use the vi editor to avoid this problem, or change the field-separator character with the **J** option. (For ease of reference, this discussion assumes you use the vi editor to modify the **/etc/sendmail.cf** file.)

After changing any information in the **/etc/sendmail.cf** file, you must compile the file into a database format that the **sendmail** command can read. See the next section, "Compiling the sendmail.cf File."

Compiling the sendmail.cf File

Note: This function is available in AIX Version 3.2.5 and AIX Version 4.1 only.

To compile the **/etc/sendmail.cf** file into a database format that the **sendmail** command can read, enter the command:

```
/usr/sbin/sendmail -bz
```

This creates the file **/etc/sendmail.cfDB**. The **/etc/sendmail.cfDB** file contains the database version of the configuration information. If you want the changes to take effect immediately, make the **sendmail** daemon reread the configuration information.

Making the sendmail Daemon Reread the Configuration Information

After you have made changes to the **sendmail.cf** file, instruct the daemon to reread the file. If you started the **sendmail** command using the **startsrc** command, enter the command:

```
refresh -s sendmail
```

Or, if you started the **sendmail** daemon using the **/usr/sbin/sendmail** command, enter the command:

```
kill -1 ‘cat /etc/sendmail.pid‘
```

Both of these commands cause the daemon to reread the **/etc/sendmail.cf** file, the **/etc/aliases** file, and the **/etc/sendmail.nl** file.

Alias Database

The alias database exists in two forms. One is a text form, maintained in the file **/etc/aliases**. The aliases are of the form:

```
name: name1, name2, ...
```

Only local names may be aliased. For example:

```
linda@cloud.ai.acme.org: linda@CS.
```

will not have the desired effect. Aliases may be continued by starting any continuation lines with a space or a tab. Blank lines and lines beginning with a pound sign (#) are comments.

The second form is processed by the ndbm or db library. This is the form that **sendmail** actually uses to resolve aliases. This technique is used to improve performance. This form is in the files:

/etc/aliases.dir and **/etc/aliases.pag**	For AIX Version 4.2 or later.
/etc/aliasesDB/DB.dir and **/etc/aliasesDB/DB.pag**	For AIX Version 3.2.5 and AIX Version 4.1. only.

Beginning with AIX Version 4.2, the control of search order is actually set by the service switch. The entry

```
OAswitch:aliases
```

is always added as the first alias entry. Also, the first alias file name without a class (for example, without "nis:" on the front) will be used as the name of the file for a "files" entry in the aliases switch. For example, if the configuration file contains:

```
OA/etc/aliases
```

and the service switch contains:

```
aliases nis files
```

aliases will first be searched in the NIS database, then in **/etc/aliases**.

Rebuilding the Alias Database

The DB or DBM version of the database may be rebuilt explicitly by executing the command:

```
newaliases
```

This is equivalent to giving **sendmail** the **-bi** flag:

```
/usr/sbin/sendmail -bi
```

If the RebuildAliases (old **D**) option is specified in the configuration, **sendmail** will rebuild the alias database automatically if possible when it is out of date. Auto-rebuild can be dangerous on heavily loaded machines with large alias files. If it might take more than the rebuild timeout (option AliasWait, old **a**, which is normally five minutes) to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

If you have multiple aliases databases specified, the **-bi** flag rebuilds all the database types. II understands, for example, it can rebuild NDBM databases, but not NIS databases.

Potential Problems with the Alias Database

There are a number of problems that can occur with the alias database. They all result from a **sendmail** process accessing the DBM version while it is only partially built. This can happen under two circumstances: One process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

Sendmail has three techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone aborting the process leaving a partially rebuilt database. Second, it locks the database source file during the rebuild, but that may not work over NFS or if the file is unwritable. Third, at the end of the rebuild, it adds an alias of the form:

```
@: @
```

(which is not normally legal). Before **sendmail** will access the database, it checks to ensure that this entry exists.

List Owners

If an error occurs on sending to a certain address, for example,"x", sendmail will look for an alias of the form "owner-x" to receive the errors. This is typically useful for a mailing list where the submitter of the list has no control over the maintenance of the list itself. In this case, the list maintainer would be the owner of the list. For example:

```
unix-wizards: linda@paintbox, wnj@monet, nosuchuser,
  sam@matisse
owner-unix-wizards: unix-wizards-request
unix-wizards-request: linda@paintbox
```

would cause `linda@paintbox` to get the error that will occur when someone sends to unix-wizards due to the inclusion of `nosuchuser` on the list.

List owners also cause the envelope sender address to be modified. The contents of the owner alias are used if they point to a single user. Otherwise, the name of the alias itself is used. For this reason, and to conform to Internet conventions, the "owner-" address normally points at the "-request" address; this causes messages to go out with the typical Internet convention of using "list-request" as the return address.

Per-User Forwarding (.forward Files)

As an alternative to the alias database, users may put a file with the name ".forward" in their home directory. If this file exists, **sendmail** redirects mail for that user to the list of addresses listed in the **.forward** file. For example, if the home directory for user "kenly" has a **.forward** file with contents:

```
kenly@ernie
joel@renoir
```

then any mail arriving for "kenly" will be redirected to the specified accounts.

Actually, the configuration file defines a sequence of filenames to check. By default, this is the user's **.forward** file, but can be defined to be more general using the **J** option. If you change this, you will have to inform your user base of the change.

IDENT Protocol Support

Beginning with AIX Version 4.2, UCB sendmail 8.7 supports the IDENT protocol as defined in RFC 1413. Although this enhances identification of the author of an e-mail message by doing a ‘‘callback’’ to the originating system to include the owner of a particular TCP connection in the audit trail, it is in no sense perfect; a determined forger can easily violate the security of the IDENT protocol.

> **Note:** The AIX operating system does not support the IDENT protocol. The IDENT timeout is set to zero (0) in the **/etc/sendmail.cf** file to disable IDENT. Modify your **sendmail.cf** file and set IDENT timeout if you wish to enable IDENT.

The following description is excerpted from RFC 1413:

6. Security Considerations

The information returned by this protocol is at most as trustworthy as the host providing it OR the organization operating the host. For example, a PC in an open lab has few if any controls on it to prevent a user from having this protocol return any identifier the user wants. Likewise, if the host has been compromised the information returned may be completely erroneous and misleading.

The Identification Protocol is not intended as an authorization or access control protocol. At best, it provides some additional auditing information with respect to TCP connections. At worst, it can provide misleading, incorrect, or maliciously incorrect information.

The use of the information returned by this protocol for other than auditing is strongly discouraged. Specifically, using Identification Protocol information to make access control decisions, either as the primary method (that is, no other checks) or as an adjunct to other methods may result in a weakening of normal host security.

An Identification server may reveal information about users, entities, objects or processes which might normally be considered private. An Identification server provides service which is a rough analog of the CallerID services provided by some phone companies and many of the same privacy considerations and arguments that apply to the CallerID service apply to Identification. If you wouldn’t run a "finger" server due to privacy considerations you may not want to run this protocol.

Tuning

Beginning with AIX Version 4.2, there are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in **sendmail.cf**. For example,the line "O Time-out.queuereturn=5d" sets option "Timeout.queuereturn" to the value "5d" (five days).

Most of these options have appropriate defaults for most sites. However, sites having very high mail loads may find they need to tune them as appropriate for their mail load. In particular, sites experiencing a large number of small messages, many of which are delivered to many recipients, may find that they need to adjust the parameters dealing with queue priorities.

All versions of **sendmail** prior to version 8.7 had single-character option names. As of version 8.7, options have long (multi-character) names. Although old short names are still accepted, most new options do not have short equivalents.

This section only describes the options you are most likely to want to tweak; read <<add pointer to section 5 here>> for more details.

Timeouts

All time intervals are set using a scaled syntax. For example, "10m" represents ten minutes, whereas "2h30m" represents two and a half hours. The full set of scales is:

s seconds

m minutes

h hours

d days

w weeks

Read Timeouts

Beginning with AIX Version 4.2, timeouts all have option names "Time-out.suboption". The recognized suboptions, their default values, and the minimum values allowed by RFC 1123 section 5.3.2 are :

connect The time to wait for an SMTP connection to open (the connect(2) system call) [0, unspecified]. If zero, uses the kernel default. In no case can this option extend the timeout longer than the kernel provides, but it can shorten it. This is to get around kernels that provide an extremely long connection timeout (90 minutes in one case).

initial The wait for the initial 220 greeting message [5m, 5m].

helo The wait for a reply from a HELO or EHLO command [5m, unspecified]. This may require a host name lookup, so five minutes is probably a reasonable minimum.

mail The wait for a reply from a MAIL command [10m, 5m].

rcpt The wait for a reply from a RCPT command [1h, 5m]. This should be long because it could be pointing at a list that takes a long time to expand (see below).

datainit The wait for a reply from a DATA command [5m, 2m].

datablock The wait for reading a data block (that is, the body of the message). [1h, 3m]. This should be long because it also applies to programs piping input to **sendmail** which have no guarantee of promptness.

datafinal The wait for a reply from the dot terminating a message. [1h,10m]. If this is shorter than the time actually needed for the receiver to deliver the message, duplicates will be generated. This is discussed in RFC1047.

rset The wait for a reply from a RSET command [5m, unspecified].

quit The wait for a reply from a QUIT command [2m, unspecified].

misc The wait for a reply from miscellaneous (but short) commands such as NOOP (no-operation) and VERB (go into verbose mode). [2m, unspecified].

command In server SMTP, the time to wait for another command. [1h, 5m].

ident The timeout waiting for a reply to an IDENT query [30s11, unspecified].

For compatibility with old configuration files, if no suboption is specified, all the timeouts marked with **-** are set to the indicated value.

Timeout Options (for AIX Version 3.2.5 and AIX Version 4.1 Only)

The **sendmail** command process can produce many types of timeouts. The time values for the different timeouts can be changed in the **/etc/sendmail.cf** file. You can set the timeout options for:

For No Data Received	The **sendmail** command times-out if it receives no data for a certain time, when reading standard input or from a remote SMTP protocol. The default configuration file sets this value to 5 minutes. If you need to change the time-out value, change the `r` option in the configuration file: `Or`*`TimeValue`* *TimeValue* is the length of time the **sendmail** command waits before timing out.
When a Message Cannot be Sent	After a message has been in the queue for a period of time, the **sendmail** command notifies the sender that the message could not be sent. The default time out is 3 days. To set this time out with the `T` option in the **/etc/sendmail.cf** configuration file, enter: `OT`*`TimeValue`* *TimeValue* is the length of time the **sendmail** command leaves a message in the queue before sending it back.

Message Timeouts

After sitting in the queue for a few days, a message will time out. This is to ensure that at least the sender is aware of the inability to send a message. The timeout is typically set to five days. It is sometimes considered convenient to also send a warning message if the message is in the queue longer than a few hours (assuming you normally have good connectivity; if your messages normally took several hours to send, you would not want to do this because it would not be an unusual event). These timeouts are set using the Timeout.queuereturn and Timeout.queuewarn options in the configuration file (previously both were set using the **T** option).

Since these options are global, and since you cannot know how long another host outside your domain will be down, a five-day timeout is recommended. This allows a recipient to fix the problem even if it occurs at the beginning of a long weekend. RFC 1123 section 5.3.1.1 says that this parameter should be‘‘at least 4-5 days’’.

The Timeout.queuewarn value can be piggybacked on the **T** option by indicating a time after which a warning message should be sent; the two timeouts are separated by a slash. For example, the line:

```
OT5d/4h
```

causes e-mail to fail after five days, but a warning message will be sent after four hours. This should be large enough that the message will have been tried several times.

Queue interval

The argument to the **-q** flag specifies how often a subdaemon will run the queue. This is typically set to between fifteen minutes and one hour. RFC 1123, section 5.3.1.1 recommends this be at least 30 minutes.

Forking During Queue Runs

By setting the ForkEachJob (**Y**) option, **sendmail** will fork before each individual message while running the queue. This will prevent **sendmail** from consuming large amounts of memory, so it may be useful in memory-poor environments. However, if the ForkEachJob option is not set, **sendmail** will keep track of hosts that are down during a queue run, which can improve performance dramatically.

If the ForkEachJob option is set, **sendmail** cannot use connection caching.

Queue Priorities

Every message is assigned a priority when it is first instantiated, consisting of the message size (in bytes) offset by the message class (which is determined from the Precedence: header) times the "work class factor"and the number of recipients times the "work recipient factor." The priority is used to order the queue. Higher numbers for the priority mean that the message will be processed later when running the queue.

The message size is included so that large messages are penalized relative to small messages. The message class allows users to send "high priority" messages by including a "Precedence:" field in their message; the value of this field is looked up in the P lines of the configuration file. Since the number of recipients affects the amount of load a message presents to the system, this is also included into the priority.

The recipient and class factors can be set in the configuration file using the RecipientFactor (**y**) and ClassFactor (**z**) options respectively. They default to 30000 (for the recipient factor) and 1800 (for the class factor). The initial priority is:

```
pri = msgsize - (class times bold ClassFactor) + (nrcpt times bold
RecipientFactor)
```

(Remember that higher values for this parameter actually mean that the job will be treated with lower priority.)

The priority of a job can also be adjusted each time it is processed (that is, each time an attempt is made to deliver it) using the "work time factor," set by the RetryFactor(**Z**) option. This is added to the priority, so it normally decreases the precedence of the job, on the grounds that jobs that have failed many times will tend to fail again in the future. The RetryFactor option defaults to 90000.

Load Limiting

Sendmail can be asked to queue (but not deliver) mail if the system load average gets too high using the QueueLA **(x**) option. When the load average exceeds the value of the QueueLA option, the delivery mode is set to **q** (queue only) if the QueueFactor (q) option divided by the difference in the current load average and the QueueLA option plus one exceeds the priority of the message; that is, the message is queued if:

```
pri > { bold QueueFactor } over { LA - { bold QueueLA } + 1 }
```

The QueueFactor option defaults to 600000, so each point of load average is worth 600000 priority points (as described above).

For drastic cases, the RefuseLA (X) option defines a load average at which **sendmail** will refuse to accept network connections. Locally generated mail (including incoming UUCP mail) is still accepted.

Delivery Mode

There are a number of delivery modes that **sendmail** can operate in, set by the DeliveryMode (**d**) configuration option. These modes specify how quickly mail will be delivered. Legal modes are:

i Deliver interactively (synchronously)

b Deliver in background (asynchronously)

q Queue only (do not deliver)

d Defer delivery attempts (do not deliver). The **d** mode is available beginning with AIX Version 4.2.

There are tradeoffs. Mode **i** gives the sender the quickest feedback, but may slow down some mailers and is hardly ever necessary. Mode **b** delivers promptly, but can cause large numbers of processes if you have a mailer that takes a long time to deliver a message. Mode **q** minimizes the load on your machine, but means that delivery may be delayed for up to the queue interval. Mode **d** is identical to mode **q** except that it also prevents all the early map lookups from working; it is intended for "dial on demand’’ sites where DNS lookups might be very expensive. Some simple error messages (for example, `host unknown during the SMTP protocol`) will be delayed using this mode. Mode **b** is the default.

If you run in mode **q** (queue only), **d** (defer), or **b** (deliver in background), **sendmail** will not expand aliases and follow **.forward** files upon initial receipt of the mail. This speeds up the response to RCPT commands. Mode **i** cannot be used by the SMTP server.

Log Level

The level of logging can be set for **sendmail**. The default using a standard configuration table is level 9. The levels for AIX Version 4.2 (or later) are as follows:

0 No logging.

1 Serious system failures and potential security problems.

2 Lost communications (network problems) and protocol failures.

3 Other serious failures.

4 Minor failures.

5 Message collection statistics.

6 Creation of error messages, VRFY and EXPN commands.

7 Delivery failures (host or user unknown, etc.).

8 Successful deliveries and alias database rebuilds.

9 Messages being deferred (due to a host being down, etc.).

10 Database expansion (alias, forward,and userdb lookups).

20 Logs attempts to run locked queue files. These are not errors, but can be useful to note if your queue appears to be clogged.

30 Lost locks (only if using **lockf** instead of **flock**).

The following log levels apply to AIX Version 3.2.5 and AIX Version 4.1 only:

0 Logs major activities only (building a configuration file or creating an alias database).

1 Logs major problems only.

2 Logs message collections and unsuccessful deliveries.

3 Logs successful deliveries.

4 Logs deferred messages (for example, due to a host being down).

5 Logs messages that are placed in the queue (normal event).

6 Logs unusual but nonharmful incidents (for example, trying to process a locked file).

9 Logs the internal queue ID to external message ID mappings. This can be useful for tracing a message as it travels between several hosts. (This is the default.)

12 Logs several messages useful when debugging.

22 Logs verbose information regarding the queue and other activities.

Additionally, values above 64 are reserved for extremely verbose debugging output.

File Modes

The modes used for files depend on what functionality you want and the level of security you require.

The database that **sendmail** actually uses is represented by the following two files:

/etc/aliases.dir and **/etc/aliases.pag**	For AIX Version 4.2 or later.
/etc/aliasesDB/DB.dir and **/etc/aliasesDB/DB.pag**	For AIX Version 3.2.5 and AIX Version 4.1. only.

The mode on these files should match the mode of **/etc/aliases**. If aliases is writable and the files for AIX Version 4.2 (or later) are not, users will be unable to reflect their desired changes through to the actual database. However, if aliases is read-only and the AIX Version 4.2 (or later) DBM files are writable, a slightly sophisticated user can arrange to steal mail anyway.

If your AIX Version 4.2 (or later) DBM files are not writable,or you do not have auto-rebuild enabled (with the AutoRebuildAliases option), then you must be careful to reconstruct the alias database each time you change the text version:

```
newaliases
```

If this step is ignored or forgotten, any intended changes will be lost.

Connection Caching

Beginning with AIX Version 4.2, when processing the queue, **sendmail** will try to keep the last few open connections open to avoid startup and shutdown costs. This only applies to IPC connections.

When trying to open a connection, the cache is first searched. If an open connection is found, it is probed to see if it is still active by sending a NOOP command. It is not an error if this fails; instead, the connection is closed and reopened.

Two parameters control the connection cache. The ConnectionCacheSize (k) option defines the number of simultaneous open connections that will be permitted. If it is set to zero, connections will be closed as quickly as possible. The default is one. This should be set as appropriate for your system size; it will limit the amount of system resources that **sendmail** will use during queue runs. Never set this higher than 4.

The ConnectionCacheTimeout (K) option specifies the maximum time that any cached connection will be permitted to idle. When the idle time exceeds this value, the connection is closed. This number should be small (under ten minutes) to prevent you from grabbing too many resources from other hosts. The default is five minutes.

Name Server Access

If you want machine exchange (MX) support, you must be using Domain Name Services (DNS).

The ResolverOptions(I) option allows you to tweak name server options. The command line takes a series of flags as documented inresolver(3) (with the leading "RES_" deleted). Each can be preceded by an optional ‘+’ or ‘-’. For example, the line:

```
O ResolverOptions=+AAONLY -DNSRCH
```

turns on the AAONLY (Accept Authoritative Answers only) and turns off the DNSRCH (search the domain path) options. Most resolver libraries default DNSRCH, DEFNAMES, and RECURSE flags on and all others off. You can also include "HasWildcardMX" to specify that there is a wildcard MX record matching your domain; this turns off MX matching when canonifying names, which can lead to inappropriate canonifications.

Moving the Per-User Forward Files

Note: This function is available beginning in AIX Version 4.2 only.

Some sites mount each user's home directory from a local disk on their workstation, so that local access is fast. However, the result is that **.forward** file lookups are slow. In some cases, mail can even be delivered on machines inappropriately because of a file server being down. The performance can be especially bad if you run the automounter.

The ForwardPath (**J**) option allows you to set a path of forward files. For example, the config file line:

```
O ForwardPath=/var/forward/$u:$z/.forward.$w
```

would first look for a file with the same name as the user's login in **/var/forward**. If that is not found (or is inaccessible), the file ".forward.machinename" in the user's home directory is searched.

If you create a directory such as **/var/forward**, it should be mode 1777 (that is, the sticky bit should be set). Users should create the files mode 644.

Free Space

Note: This function is available beginning in AIX Version 4.2 only.

On systems that have one of the system calls in the statfs(2) family (including **statvfs** and **ustat**), you can specify a minimum number of free blocks on the queue filesystem using the MinFreeBlocks (**b**) option. If there are fewer than the indicated number of blocks free on the filesystem on which the queue is mounted, the SMTP server will reject mail with the 452 error code. This invites the SMTP client to try again later.

Attention: Be careful not to set this option too high; it can cause rejection of e-mail when that mail would be processed without difficulty.

Maximum Message Size

Note: This function is available beginning in AIX Version 4.2 only.

To avoid overflowing your system with a large message, the MaxMessageSize option can set an absolute limit on the size of any one message. This will be advertised in the ESMTP dialogue and checked during message collection.

Privacy Flags

Note: This function is available beginning in AIX Version 4.2 only.

The PrivacyOptions (**p**) option allows you to set certain "privacy" flags. Actually, many of them don’t give you any extra privacy, rather just insisting that client SMTP servers use the HELO command before using certain commands or adding extra headers to indicate possible security violations.

The option takes a series of flag names; the final privacy is the inclusive or of those flags. For example:

```
O PrivacyOptions=needmailhelo, noexpn
```

insists that the HELO or EHLO command be used before a MAIL command is accepted and disables the EXPN command.

The flags are detailed in RFC 1123 S 5.1.6.

Send to Me Too

Normally, **sendmail** deletes the (envelope) sender from any list expansions. For example, if "linda" sends to a list that contains "linda" as one of the members, she will not get a copy of the message. If the **-m** (me too) command line flag, or if the MeToo (**m**) option is set in the configuration file, this behavior is suppressed.

C and F -- Define Classes

Classes of phrases may be defined to match on the left hand side of rewrite rules, where a "phrase" is a sequence of characters that do not contain space characters. For example, a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes are named as a single letter or a word in {braces}. Class names beginning with lowercase letters and special characters are reserved for system use. Classes defined in config files may be given names from the set of uppercase letters for short names or beginning with an uppercase letter for long names.

```
Ccphrase1 phrase2...
Fcfile
```

The first form defines the class c to match any of the named words. It is permissible to split them among multiple lines; for example, the two forms:

```
CHmonet ucbmonet
```

and

```
CHmonet
CHucbmonet
```

are equivalent. The ‘‘F’’ form reads the elements of the class c from the named file.

Elements of classes can be accessed in rules using $= or $~. The $~ (match entries not in class) only matches a single word; multi-word entries in the class are ignored in this context.

The class $=w is set to be the set of all names this host is known by. This can be used to match local hostnames.

The class $=k is set to be the same as $k, that is, the UUCP node name.

The class $=m is set to the set of domains by which this host is known, initially just $m.

The class $=t is set to the set of trusted users by the T configuration line. If you want to read trusted users from a file, use **Ft/file/name**.

The class $=n can be set to the set of MIME body types that can never be eight to seven bit encoded. It defaults to "multipart/signed". Message types "message/*" and "multipart/*" are never encoded directly. Multipart messages are always handled recursively. The handling of message/* messages are controlled by class $=s. The class $=e contains the Content-Transfer-Encodings that can be 8->7 bit encoded. It is predefined to contain "7bit", "8bit", and "binary". The class $=s contains the set of subtypes of message that can be treated recursively. By default it contains only "rfc822". Other "message/*" types cannot be 8->7 bit encoded. If a message containing eight-bit data is sent to a seven-bit host, and that message cannot be encoded into seven bits, it will be stripped to 7 bits.

Beginning with AIX Version 4.2, the three classes $=U, $=Y, and $=Z are defined to describe the hosts requiring the use of a uucp mailer. Specifically, $=U should contain all hosts requiring the uucp-old mailer. $=Y should contain all hosts requiring the uucp-new mailer. Finally, $=Z should contain all hosts requiring the uucp-uudom mailer. Each uucp host should belong to one of these classes.

Sendmail can be compiled to allow a scanf(3) string on the F line. This lets you do simplistic parsing of text files. For example, to read all the user names in your system **/etc/passwd** file into a class, use:

```
FL/etc/passwd %[^:]
```

which reads every line up to the first colon.

Changing the Host Name

Cw contains all the possible names for the local host. It defines aliases. **Cw** specifies the name and all aliases for your host system. If your system uses different names for two different network connections, enter both names as part of the host name class. If you do not define both names, mail sent to the undefined name is returned to the sender.

```
CwCw alias aliasn...
```

By default, the **sendmail** command reads what has been set with the **hostname** command and uses it to initialize the host and domain name macros and classes. Change the configuration file macros only if you want the sendmail host and domain names to be different from those set by the **hostname** command.

To change the host name:

1. Enter the command:

   ```
   vi /etc/sendmail.cf
   ```

2. Find the lines beginning with `Dj, Dn, and Dw`. `Dj, Dn, and Dw` override the host and domain names set with "hostname".

3. Replace `Dj, Dn, and Dw` with the new hostname information. For example, if your hostname is `brown.newyork.abc.com`, and you have one alias, `brown2`, enter:
4. Save the file and exit the editor.

Creating a Class Using a File

To define a class whose members are listed in an external file (one member per line), use a control line that begins with the letter `F`. The syntax for the `F` class definition is:

```
FClass FileName [Format]
```

`Class` is the name of the class that matches any of the words listed in *`FileName`*. *`Filename`* is the full path name of file (for convenience, you may wish to put the file in the **/etc** directory). *`Format`* is an optional **scanf** subroutine format specifier that indicates the format of the elements of the class in *FileName*. The *`Format`* specifier can contain only one conversion specification.

M -- Define Mailer

Programs and interfaces to mailers are defined in this line. The format is:

```
Mname, {field=value}*
```

where name is the name of the mailer (used internally only) and the "field=name" pairs define attributes of the mailer. Fields are:

Path The pathname of the mailer

Flags Special flags for this mailer

Sender Rewrite set(s) for sender addresses

Recipient Rewrite set(s) for recipient addresses

Argv An argument vector to pass to this mailer

Eol The end-of-line string for this maile

Maxsize The maximum message length to this mailer

Linelimit The maximum line length in the message body (field available beginning with AIX Version 4.2)

Directory The working directory for the mailer (field available beginning with AIX Version 4.2)

Userid The default user and group id to run as (field available beginning with AIX Version 4.2)

Nice The nice(2) increment for the mailer (field available beginning with AIX Version 4.2)

Charset The default character set for 8-bit characters (field available beginning with AIX Version 4.2)

Type The MTS type information (used for error messages) (field available beginning with AIX Version 4.2)

Only the first character of the field name is checked.

The flags in the following list may be set in the mailer description. Any other flags may be used freely to conditionally assign headers to messages destined for particular mailers. Flags marked with - are not interpreted by the **sendmail** binary; these are conventionally used to correlate to the flags portion of the H line. Flags marked with = apply to the mailers for the sender address rather than the usual recipient mailers.

a Run Extended SMTP (ESMTP) protocol (defined in RFCs 1651, 1652, and 1653). This flag defaults on if the SMTP greeting message includes the word "ESMTP". (Flag available beginning with AIX Version 4.2)

A Look up the user part of the address in the alias database. Normally this is only set for local mailers. (Flag available beginning with AIX Version 4.2)

b Force a blank line on the end of a message. This is intended to work around some versions of **/bin/mail** that require a blank line, but do not provide it themselves. It would not normally be used on network mail. (Flag available beginning with AIX Version 4.2)

c Do not include comments in addresses. This should only be used if you have to work around a remote mailer that gets confused by comments. This strips addresses of the form "Phrase <address>" or "address (Comment)" down to just "address". (Flag available beginning with AIX Version 4.2)

C= If mail is received from a mailer with this flag set, any addresses in the header that do not have an at sign ("@") after being rewritten by ruleset three will have the "@domain" clause from the sender envelope address tacked on. This allows mail with headers of the form:

```
From: usera@hosta
To: userb@hostb, userc
```

to be rewritten automatically (although *not* reliably) as:

```
From: usera@hosta
To: userb@hostb, userc@hosta
```

d Do not include angle brackets around route-address syntax addresses. This is useful on mailers that are going to pass addresses to a shell that might interpret angle brackets as I/O redirection.

D- This mailer wants a "Date:" header line.

e This mailer is expensive to connect to, so try to avoid connecting normally. Any necessary connection will occur during a queue run.

E Escape lines beginning with "From" in the message with a ‘>’ sign.

f The mailer wants a **-f** from flag, but only if this is a network forward operation (that is, the mailer will give an error if the executing user does not have special permissions).

F- This mailer wants a "From:" header line.

g Normally, **sendmail** sends internally generated error messages using the null return address as required by RFC 1123. However, some mailers don’t accept a null return address. If necessary, you can set the **g** flag to prevent **sendmail** from obeying the standards; error messages will be sent as from the MAILER-DAEMON (actually, the value of the $n macro). (Flag available beginning with AIX Version 4.2)

h Uppercase should be preserved in host names for this mailer.

I This mailer will be speaking SMTP to another **sendmail,** as such it can use special protocol features. This option is not required (i.e., if this option is omitted the transmission will still operate successfully, although perhaps not as efficiently as possible).

j Do User Database rewriting on recipients as well as senders.

k Normally when **sendmail** connects to a host via SMTP, it checks to make sure that this isn’t accidentally the same host name as might happen if **sendmail** is misconfigured or if a long-haul network interface is set in loopback mode. This flag disables the loopback check. It should only be used under very unusual circumstances. (Flag available beginning with AIX Version 4.2)

K Currently unimplemented. Reserved for chunking.

l This mailer is local (that is, final delivery will be performed).

L Limit the line lengths as specified in RFC821. This deprecated option should be replaced by the L= mail declaration. For historic reasons, the **L** flag also sets the 7 flag.

m This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the argv part of the mailer definition, that field will be repeated as necessary for all qualifying users.

M- This mailer wants a "Message-Id:" header line.

n Do not insert a UNIX-style "From" line on the front of the message.

o Always run as the owner of the recipient mailbox. Normally **sendmail** runs as the sender for locally generated mail or as "daemon" (actually, the user specified in the **u** option) when delivering network mail. The normal behavior is required by most local mailers, which will not allow the envelope sender address to be set unless the mailer is running as daemon. This flag is ignored if the **S** flag is set. (Flag available beginning with AIX Version 4.2)

p Use the route-addr style reverse-path in the SMTP "MAIL FROM:" command rather than just the return address; although this is required in RFC821 section 3.1, many hosts do not process reverse-paths properly. Reverse-paths are officially discouraged by RFC 1123.

P- This mailer wants a "Return-Path:" line.

q When an address that resolves to this mailer is verified (SMTP VRFY command), generate 250 responses instead of 252 responses. This will imply that the address is local.

r Same as **f**, but sends an **-r** flag.

R Open SMTP connections from a "secure" port. Secure ports aren't secure except on UNIX machines, so it is unclear that this adds anything.

s Strip quote characters (" and \) off the address before calling the mailer.

S Don't reset the userid before calling the mailer. This would be used in a secure environment where **sendmail** ran as root. This could be used to avoid forged addresses. If the **U=** field is also specified, this flag causes the userid to always be set to that user and group (instead of leaving it as root).

u Uppercase should be preserved in user names for this mailer.

U This mailer wants UUCP-style "From" lines with the "remote from <host>" on the end.

w The user must have a valid account on this machine (getpwnam must succeed). If not, the mail is bounced. This is required to get ".forward" capability. (Flag available beginning with AIX Version 4.2)

x- This mailer wants a "Full-Name:" header line.

X This mailer wants to use the hidden dot algorithm as specified in RFC821; basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This ensures that lines in the message containing a dot will not terminate the message prematurely.

z Run Local Mail Transfer Protocol (LMTP) between **sendmail** and the local mailer. This is a variant on SMTP defined in RFC 2033 that is specially designed for delivery to a local mailbox.

0 Don't look up Mx records for hosts via SMTP.

3 Extend the list of characters converted to `=XX` notation when converting to `Quoted-Printable` to include those that don't map cleanly between ASCII and EBCDIC. Useful if you have IBM mainframes on site.

5 If no aliases are found for this address, pass the address through ruleset 5 for possible alternate resolution. This is intended to forward the mail to an alternate delivery spot. (Flag available beginning with AIX Version 4.2)

7 Strip all output to seven bits. This is the default if the **L** flag is set. Note that clearing this option is not sufficient to get full eight-bit data passed through **sendmail**. If the 7 option is set, this is essentially always set, since the eighth bit was stripped on input. Note that this option will only impact messages that didn't have 8->7 bit MIME conversions performed. (Flag available beginning with AIX Version 4.2)

8 If set, it is acceptable to send eight bit data to this mailer; the usual attempt to do 8->7 bit MIME conversions will be bypassed. (Flag available beginning with AIX Version 4.2)

9 If set, do limited 7->8 bit MIME conversions. These conversions are limited to text/plain data.

: Check addresses to see if they begin ":include:". If they do,convert them to the "*include*" mailer. (Flag available beginning with AIX Version 4.2)

| Check addresses to see if they begin with a '|'. If they do, convert them to the "prog" mailer. (Flag available beginning with AIX Version 4.2)

/ Check addresses to see if they begin with a '/'. If they do, convert them to the "*file*" mailer. (Flag available beginning with AIX Version 4.2)

@ Look up addresses in the user database. (Flag available beginning with AIX Version 4.2)

Note: Configuration files prior to level 6 assume the 'A', 'w', '5', ':', '|', '/', and '@' options on the mailer named "local".

The mailer with the special name "error" can be used to generate a user error. The (optional) host field is an exit status to be returned, and the user field is a message to be printed. The exit status may be numeric or one of the values USAGE, NOUSER, NOHOST, UNAVAILABLE, SOFTWARE, TEMPFAIL, PROTOCOL, or CONFIG to return the corresponding EX_ exit code. For example, the entry:

```
$#error $@ NOHOST $: Host unknown in this domain
```

on the RHS of a rule will cause the specified error to be generated and the "Host unknown" exit status to be returned if the LHS matches. It is always available for use in O, S, and check_ ... rulesets and it cannnot be defined with **M** commands.

The mailer named "local" must be defined in every configuration file. This is used to deliver local mail, and is treated specially in several ways. Additionally, three other mailers named "prog", "*file*",and "*include*" may be defined to tune the delivery of messages to programs, files, and :include: lists respectively. They default to:

```
Mprog, P=/bin/sh, F=lsoDq9, T=DNS/RFC822/X-Unix, A=sh -c $u
M*file*, P=[FILE], F=lsDFMPEuq9, T=DNS/RFC822/X-Unix, A=FILE $u
M*include*, P=/dev/null, F=su, A=INCLUDE $u
```

The Sender and Recipient rewrite sets may either be a simple ruleset id or may be two ids separated by a slash If so, the first rewrite set is applied to envelope addresses, and the second is applied to headers. Setting any value to zero disables the corresponding mailer-specific rewriting.

The `Directory` field is a path of directories to try. For example, the definition `D=$z:/` tries to execute the recipient's home directory, but if that is not available, it tries to execute in the root of the filesystem. Use this on the **prog** mailer only, since some shells (e.g., **csh**) do not execute if they cannot read the home directory. Since the queue directory usually cannot be read by unauthorized users, **csh** scripts can fail if they are used as recipients.

The `Userid` field specifies the default user and group id to run. It overrides the **DefaultUser** option *q.v.* If the **S** mailer flag is also specified, the user and group id will run in all circumstances. Use the form *user:group* to set both the user and group id. Either of these variables may be an interger or a symbolic name that is looked up in the **passwd** and **group** files respectively.

The `Charset` field is used when converting a message to MIME. It is the character set used in the Content-Type: *header*. If it is not set, the **DefaultCharset** option is used. If the **DefaultCharset** is not set, the value `unknown-8bit` is used. The **Charset** field applies to the *sender's* mailer; *not the recipient's* mailer. For example: if the envelope sender address is on the local network and the recipient is on an external network, the character set is set from the `Charset=` field for the local network mailer, not the external network mailer.

The `Type` field sets the type of information used in MIME error messages (as defined by RFC 1984). It contains three values that are separated by slashes: the MTA type (a description of how hosts are named), address type (a description of e-mail addresses), and diagnostic type (a description of error diagnostic codes). Each must be a registered value or begin with `X-`. The default is `dns/rfc822/smtp`.

Mailer Specifications Examples

1. To specify a local delivery mailer enter:

   ```
   Mlocal, P=/usr/bin/bellmail, F=lsDFMmn, S=10, R=20, A=mail $u
   ```

   The mailer is called `local`. Its path name is `/usr/bin/bellmail`. The mailer uses the following flags:

`l`	Specifies local delivery.
`s`	Strips quotation marks from addresses.
`DFM`	Requires `Date:`, `From:`, and `Message-ID:` fields.
`m`	Delivers to multiple users.
`n`	Does not need an operating system `From` line at the start of the message.

   Rule set 10 should be applied to sender addresses in the message. Rule set 20 should be applied to recipient addresses. Additional information sent to the mailer in the `A` field is the word *mail* and words containing the recipient's name.

H -- Define Header

The format of the header lines that **sendmail** inserts into the message are defined by the **H** line. The syntax of this line is:

```
H[?mflags?]hname: htemplate
```

Continuation lines in this spec are reflected directly into the outgoing message. The htemplate is macro expanded before insertion into the message. If the mflags (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input, it is reflected to the output regardless of these flags.

Some headers have special semantics that will be described later.

A secondary syntax allows validation of headers as they being read. To enable validation, use:

```
HHeader: $>Ruleset
```

The indicated *Ruleset* is called for the specified *Header*. Like other **check_*** rulesets, it can return `$#error` to reject the message or `$#discard` to discard the message. The header is treated as a structured field, so comments (in parentheses) are deleted before processing.

For example, the following configuration lines:

```
HMessage-Id: $>CheckMessageId

SCheckMessageId
R<$+@$+> $@OK
R$*      $#error $: Illegal Message-Id header
```

would refuse any message header that had a Message-Id: header of any of the following forms:

```
Message-Id: <>
Message-Id: some text
Message-Id: <legal test@domain> extra text
```

Message Headings in the sendmail.cf File

Lines in the configuration file that begin with a capital letter *H*, define the format of the headers used in messages. The format of the `H` command is:

Lines in the configuration file that begin with a capital letter *H*, define the format of the headers used in messages. The format of the `H` control line is:

```
H[?MailerFlags?]FieldName: Content
```

The variable parameters are defined as:

MailerFlags	Determines whether the `H` line is used. This parameter is optional. If you supply this parameter, surround it with ? (question marks). If the mailer requires the field defined by this control line (as indicated in the mailer definition's flags field), then the `H` control line is included when formatting the heading. Otherwise, the `H` control line is ignored.
FieldName	Contains the text displayed as the name of the field in the heading information. Typical field names include `From:`, `To:`, and `Subject:`.
Content	Defines the information that is displayed following the field name. Usually macros specify this information.

These example lines are from a typical **/etc/sendmail.cf** file:

`H?P?Return-Path: <$g>`	Defines a field called `Return-Path` that displays the content of the **$g** macro (sender address relative to the recipient). The `?P?` portion indicates this line is only used if the mailer uses the **P** flag (the mailer requires a `Return-Path` line). The header is generated only if the mailer has the indicated flag. If the header appears in the input message, it is passed through unchanged.

`HReceived: $?sfrom $s $.by $j ($v/$Z) id $i; $b`

Defines a field called `Received`. This field includes:

`$?sfrom $s $.`	Displays the text `from` followed by the content of the **$s** macro if an **s** macro is defined (sender's host name).
`by $j`	Displays the text `by` followed by the content of the **$j** macro (official name for a specific location).
`($v/$Z)`	Displays the version of the **sendmail** command (**$v**) and the version of the **/etc/sendmail.cf** file (**$Z**), set off by parentheses and separated by a slash.
`id $i;`	Displays the text `id` followed by the content of the **$i** macro (mail-queue ID of the message) and a ; (semicolon).
`$b`	Displays the current date.

O -- Set Option

For information on the options for AIX Version 4.2 or later, see "sendmail.cf File Options (for AIX Version 4.2 or Later)" .

For information on the options for AIX Version 3.2.5 and AIX Version 4.1, see "sendmail.cf File Options (for AIX Version 3.2.5 or AIX Version 4.1)" .

P -- Precedence Definitions

Values for the "Precedence:" field may be defined using the P control line. The syntax of this field is:

```
Pname=num
```

When the name is found in a "Precedence:" field, the message class is set to num. Higher numbers mean higher precedence. Numbers less than zero have the special property that if an error occurs during processing, the body of the message will not be returned; this is expected to be used for "bulk" mail such as through mailing lists. The default precedence is zero. For example, the list of default precedences is:

- Pfirst-class=0
- Pspecial-delivery=100
- Plist=-30
- Pbulk=-60
- Pjunk=-100

V -- Configuration Version Level

To provide compatibility with old configuration files, the **V** line has been added to define basic semantics of the configuration file. This is *not* intended as long term support. These compatibility features may be removed in future releases.

N.B.: configuration version *levels* are independent of configuration file version *numbers*. For example, version *number* 8.9 configuration files use version *level* 8 configurations.

"Old" configuration files are defined as version level one.

Version level two files make the following changes:

1. Host name canonification ($[... $]) appends a dot if the name is recognized. This gives the configuration file a way to determine if a match occurred. This initializes the host map with the **-a.** flag. You can reset it to anything else by declaring the map explicitly.
2. Default host name extension is consistent throughout processing. Version level one configurations turned off domain extension during certain points in processing by adding the local domain name. Version level two configurations include a trailing dot to indicate that the name is already canonical.
3. Local names that are not aliases are passed through a new distinguished ruleset five. This can be used to append a local relay. This can be prevented by resolving the local name by using the @ symbol as a prefix (e.g, @vikki). For example; something that resolves to a local mailer and a user name of vikki will be passed through ruleset five, but a user name of @vikki will have the @ prefix stripped, will not be passed through to ruleset five, but will otherwise be treated the same as the prior example. The exception is that this might be used to implement a policy where mail

sent to vikki is handled by a central hub but mail sent to vikki@localhost is delivered directly.

Version level three files allow # initiated comments on all lines. Exceptions are backslash escaped # marks and the **$#** syntax.

Version level four files are equivalent to level three files.

Version level five files change the default definition of **$w** to be the first component of the hostname.

Version level six configuration files change many of the local processing options (i.e., aliasing and matching the address beginning for the | character) to mailer flags. This allows fine grained control over the special local processing. Version level six files may also use long option names. The **ColonOkInAddr** option (which allows colons in the local part of the address) defaults to **on** in configuration files with lower version numbers. The configuration file requires additional "intelligence" to properly handle the RFC 822 group construct.

Version level seven configuration files use new option names to replace old macros.

$e became	**SmtpGreetingMessage**
$1 became	**UnixFromLine**
$o became	**OperatorChars**

Prior to version seven, the **F=q** flag (use the return value 250 instead of 252 for SMTP VRFY commands) was assumed.

Version level eight configuration files allow **$#** on the left side of ruleset lines.

The **V** line may have an optional */vendor* variable to indicate that the configuration file uses vendor specific modifications. You may use `/Berkeley` to indicate that the file uses the Berkeley **sendmail** dialect.

K -- Key File Declaration

Note: This function is available beginning in AIX Version 4.2 only.

Special maps can be defined using the line:

```
Kmapname mapclass arguments
```

The mapname is the name by which this map is referenced in the rewrite rules. The mapclass is the name of a type of map; these are compiled in to **sendmail**. The arguments are interpreted depending on the class; typically, there would be a single argument naming the file containing the map.

Maps are referenced using the syntax:

```
$( map key $@ arguments $: default $)
```

where either or both of the arguments or default portion may be omitted. The $@ arguments may appear more than once. The indicated key and arguments are passed to the appropriate mapping function. If it returns a value, it replaces the input. If it does not return a value and the default is specified, the default replaces the input. Otherwise, the input is unchanged.

During replacement of either a map value or default, the string "%n" (where n is a digit) is replaced by the corresponding argument. Argument zero is always the database key. For example, the rule:

```
R$- ! $+    $: $(uucp $1 $@ $2 $: %1 @ %0 . UUCP
$)
```

looks up the UUCP name in a (user-defined) UUCP map. If not found, it turns it into ".UUCP" form. The database might contain records like:

```
decvax          %1@
%0.DEC.COM
research        %1@%0.ATT.COM
```

Note: The default clauses never do this mapping.

The builtin map with both name and class "host" is the host name canonicalization lookup. Thus, the syntax:

`$(host` *`hostname`*`$)`

is equivalent to:

`$[`*`hostname`*`$]`

There are many defined classes.

dbm Database lookups using the ndbm(3) library. **Sendmail** must be compiled with NDBM defined.

nis NIS lookups. **Sendmail** must be compiled with NIS defined.

ldapx LDAP X500 directory lookups. **Sendmail** must be compiled with LDAPMAP defined. The map supports most of the standard arguments and command line arguments of the **ldapsearch** program.

text Text file lookups. The format of the text file is defined by the **-k** (key field number), **-v** (value field number), and **-z** (field delimiter) flags.

stab Internal symbol table lookups. Used internally for aliasing.

implicit Really should be called "alias." This is used to get the default lookups for alias files, and is the default if no class is specified for alias files.

user Looks up users using getpwnam(3). The **-v** flag can be used to specify the name of the field to return (although this is normally used only to check the existence of a user).

host Canonifies host domain names. Given a host name, it calls the name server to find the canonical name for that host.

bestmx Returns the best MX record for a host name given as the key. The current machine is always preferred. For example, if the curent machine is one of the hosts listed as the lowest preference MX record, it will be guaranteed to be returned. This can be used to find out if this machine is the target for an MX record and mail can be accepted on that basis. If the **-z** flag is given, all MX names are returned (separated by the given delimiter).

sequence The arguments on the 'K' line are a list of maps; the resulting map searches the argument maps in order until it finds a match for the indicated key. For example, if the key definition is:

```
Kmap1 ...
Kmap1 ...
Kseqmap sequence map1 map2
```

then a lookup against "seqmap" first does a lookup in map1. If that is found, it returns immediately. Otherwise, the same key is used for map2.

switch Much like the "sequence" map except that the order of maps is determined by the service switch. The argument is the name of the service to be looked up; the values from the service switch are appended to the map name to create new map names. For example, consider the key definition:

```
Kali switch aliases
```

together with the service switch entry:

```
aliases    nis files
```

This causes a query against the map "ali" to search maps named "ali.nis" and "ali.files" in that order.

dequote Strip double quotes (") from a name. It does not strip backslashes, and will not strip quotes if the resulting string would contain unscannable syntax (that is, basic errors like unbalanced angle brackets; more sophisticated errors such as unknown hosts are not checked). The intent is for use when trying to accept mail from systems such as DECnet that routinely quote odd syntax such as:

```
"49ers::ubell"
```

A typical use is probably something like:

```
Kdequote dequote
...
R$-        $: $(dequote $1 $)
R$- $+     $: $>3 $1 $2
```

Care must be taken to prevent unexpected results; for example,

```
"|someprogram < input > output"
```

will have quotes stripped, but the result is probably not what you had intended. Fortunately, these cases are rare.

regex The map definition on the **K** line contains a regular expression. Any key input is compared to that expression using the POSIX regular expressions routines **regcomp**( ), **regerr**( ), and **regexec**( ). Refer to the documentation for those routines for more information about regular expression matching. No rewriting of the key is done if the **-m** flag is used. Without it, the key is discarded, or if **-s** is used, it is substituted by the substring matches, delimited by the **$|** or the string specified with the **-d** flag. The flags available for the map are:

- **-n** not
- **-f** case sensitive
- **-b** basic regular expressions (default is extended)
- **-s** substring match
- **-d** set the delimiter used for **-s**
- **-a** append string to key
- **-m** match only, do not replace/discard value

The **-s** flag can include an optional parameter which can be used to select the substrings in the result of the lookup. For example, `-s1,3,4`.

program The arguments on the **K** line are the pathname to a program and any initial parameters to be passed. When the map is called, the key is added to the initial parameters and the program is invoked as the default user/group id. The first line of standard output is returned as the value of the lookup. This has many potential security problems and terrible performance. It should be used only when absolutely necessary.

Most of these accept as arguments the same optional flags and a filename (or a mapname for NIS; the filename is the root of the database path, so that ".db" or some other extension appropriate for the database type will be added to get the actual database name). Known flags are:

-o Indicates that this map is optional. That is, if it cannot be opened, no error is produced, and **sendmail** will behave as if the map existed but was empty.

-N, **-O** If neither **-N** or **-O** are specified, **sendmail** uses an adaptive algorithm to decide whether or not to look for null bytes on the end of keys. It starts by trying both; if it finds any key with a null byte, it never tries again without a null byte and vice versa. If **-N** is specified, it never tries without a null byte and if **-O** is specified, it never tries with a null byte. Setting one of these can speed matches but are never necessary. If both **-N** and **-O** are specified, **sendmail** will never try any matches at all. That is, everything will appear to fail.

-ax Append the string x on successful matches. For example, the default host map appends a dot on successful matches.

-Tx Append the string x on temporary failures. For example, x would be appended if a DNS lookup returned `server failed` or an NIS lookup could not locate a server. See the **-t** flag for additional information.

-f Do not fold upper to lower case before looking up the key.

-m Match only (without replacing the value). If you only care about the existence of a key and not the value (as you might when searching the NIS map "hosts.byname" for example), this flag prevents the map from substituting the value. However, The **-a** argument is still appended on a match, and the default is still taken if the match fails.

-k*keycol* The key column name (for NIS+) or number (for text lookups).

-v*valcol* The value column name (for NIS+) or number (for text lookups).

-z*delim* The column delimiter (for text lookups). It can be a single character or one of the special strings "\n" or "\t" to indicate newline or tab respectively. If omitted entirely, the column separator is any sequence of whitespace.

-t Normally, when a map attempts to do a lookup and the server fails (e.g., **sendmail** could not contact any name server -- this is *not* the same as an entry not being found in the map), the message being processed is queued for future processing. The **-t** flag turns off this behavior, letting the temporary failure (server down) act as though it were a permanent failure (entry not found). It is particularly useful for DNS lookups, where another's misconfigured name server can cause problems on your machine. Care must be taken to avoid "bouncing" mail that would be resolved correctly if another attempt were made. A common strategy is to forward such mail to another mail server.

-s*spacesub* For the dequote map only, the character to use to replace space characters after a successful dequote.

-q Do not dequote the key before lookup.

-A When rebuilding an alias file, the **-A** flag causes duplicate entries in the text version to be merged. For example, the following two entries:

```
list:     user1,user2
list:     user3
```

would be treated as if they were the following single entry:

```
list:     user1,user2,user3
```

The dbm map appends the strings ".pag" and ".dir" to the given filename; the two db-based maps append ".db". For example, the map specification

```
Kuucp dbm -o -N /usr/lib/uucpmap
```

specifies an optional map named "uucp" of class "dbm"; it always has null bytes at the end of every string, and the data is located in **/usr/lib/uucpmap**.{dir,pag}.

Commands and Operands

C*XWord1 Word2...*	Defines the class of words that can be used to match the left-hand side of rewrite rules. Class specifiers (*X*) 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 a macro (*X*) and its associated *Value*. Macro 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 the *FileName* variable, using an optional **scanf** format specifier. The format specifier contains only one conversion specification. One class number is read for each line in the *FileName* variable.
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, 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* variable.
M*Name***,** [*Field***=***Value*]	Defines a Mail program where the *Name* variable is the name of the Mail program and *Field***=***Value* pair defines the attributes of the mailer.
O*x*[*Value*]	Sets the option to the value of *x*. If the option is a valued option, you must also specify the *Value* variable. Options may also be selected from the command line. **Note:** For valid values, see "sendmail.cf File Options (for AIX Version 4.2 or Later)" or "sendmail.cf File Options (for AIX Version 3.2.5 or AIX Version 4.1)" .

P*Name*=*Number*	Defines values for the `Precedence:` header field. When the *Name* variables found in a message's `Precedence:` field, the message's precedence is set to the *Number* variable. 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 rewrite rule. One or more tab characters separate the three fields of this command. If space characters are used as field separators, option **J** must be set. The **J** option allows spaces as well as tabs to separate the left- and right-hand sides of rewrite rules. The **J** option allows rewrite rules to be modified using an editor that replaces tabs with spaces.
S*x*	Sets the rule set currently defined to the specified number(*x*). If a rule set definition is started more than once, the new definition overwrites the old.
T*User1 User2 ...*	Defines user IDs for the system administrators. These IDs have permission to override the sender address using the **-f** flag. More than one ID can be specified per line.

sendmail.cf File Options (for AIX Version 4.2 or Later)

There are a number of global options that can be set from a configuration file. Options are represented by full words. Some can also be represented as single characters for back compatibility. The syntax of this line is:

```
O option=value
```

This sets *option* to be *value*. Note that there must be a space between the letter 'O' and the name of the option. An older version is:

```
Oovalue
```

where the option **o** is a single character. Depending on the option, value may be a string, an integer, a boolean (with legal values "t", "T", "f", or "F"; the default is TRUE), or a time interval.

The options supported are as follows. You can also use the single-character names; they are shown in brackets in the following list:

AliasFile=*spec, spec, ...*

[**A**] Specify possible alias file(s). Each *spec* should be in the format ‘‘class: file’’ where class: is optional and defaults to ‘‘implicit’’. Depending on how **sendmail** is compiled, valid classes are as follows, or if a list of specs is provided, **sendmail** searches them in order.

- **dbm** If NDBM is specified
- **nis** If NIS is specified.

AliasWait=*timeout*

[**a**] If set, wait up to *timeout* (units default to minutes) for an "@:@" entry to exist in the alias database before starting up. If it does not appear in the timeout interval, rebuild the database (if the AutoRebuildAliases option is also set) or issue a warning.

AllowBogusHELO

[no short name] If set, allow HELO SMTP commands that don’t include a host name. Setting this violates RFC 1123, section 5.2.5, but is necessary to interoperate with several SMTP clients. If there is a value, it is still checked for legitimacy. This option is valid for sendmail version 8.8 and above.

AutoRebuildAliases

[**D**] If set, rebuild the alias database if necessary and possible. If this option is not set, **sendmail** will never rebuild the alias database unless explicitly requested using **-bi**. Not recommended, since it can cause thrashing.

BlankSub=*c*

[**B**] Set the blank substitution character to *c*. Unquoted spaces in addresses are replaced by this character. Defaults to space (no change is made).

CheckAliases

[**n**] Validate the RHS of aliases when rebuilding the alias database.

CheckpointInterval=*N*

[**C**] Checkpoints the queue every *N* (default 10) addresses sent. If your system crashes during delivery to a large list, this prevents retransmission to any but the last recipients.

ClassFactor=*fact*

[**z**] The indicated factor is multiplied by the message class (determined by the Precedence: field in the user header and the P lines in the configuration file) and subtracted from the priority. Thus, messages with a higher Priority: will be favored. Defaults to 1800.

ColonOkInAddr

[no short name] If set, colons are acceptable in e-mail addresses (for example, "host:user"). If not set, colons indicate the beginning of a RFC 822 group construct ("groupname: member1, member2, ... memberN;"). Doubled colons are always acceptable ("nodename::user") and proper route-addr nesting is understood ("<@relay:user@host>"). Furthermore, this option defaults on if the configuration version level is less than 6 (for back compatibility). However, it must be off for full compatibility with RFC 822.

ConnectionCacheSize=*N*

[**k**] The maximum number of open connections that will be cached at a time. The default is one. This delays closing the current connection until either this invocation of **sendmail** needs to connect to another host or it terminates. Setting it to zero defaults to the old behavior, that is, connections are closed immediately. Since this consumes file descriptors, the connection cache should be kept small; 4 is probably a practical maximum.

ConnectionCacheTimeout=*timeout*	[**K**] The maximum amount of time a cached connection will be permitted to idle without activity. If this time is exceeded, the connection is immediately closed. This value should be small (approximately ten minutes). Before **sendmail** uses a cached connection, it always sends a RSET command to check the connection; if this fails, it reopens the connection. This keeps your end from failing if the other end times out. This option avoids using up excessive resources on the other end. The default is five minutes.
ConnectionRateThrottle=*N*	[no short name] If set to a positive value, allow no more that *N* incoming daemon connections in a one-second period. This is intended to flatten out peaks and allow the load average checking to cut in. Defaults to zero (no limits). This option is valid for sendmail version 8.8 and above.
DaemonPortOptions=*options*	[**O**] Set server SMTP options. The options are *key*=*value* pairs. Known keys are: **Port** Name/number of listening port (defaults to "smtp") **Addr** Address mask (defaults INADDR_ANY) **Family** Address family (defaults to INET) **Listen** Size of listen queue (defaults to 10) **SndBufSize** Size of TCP send buffer **RcvBufSize** Size of TCP receive buffer The Address mask may be a numeric address in dot notation or a network name.
DefaultCharSet=*charset*	[no short name] When a message that has 8-bit characters but is not in MIME format is converted to MIME (see the EightBitMode option), a character set must be included in the Content-Type: header. This character set is normally set from the Charset= field of the mailer descriptor. If that is not set, the value of this option is used. If this option is not set, the value "unknown-8bit" is used.
DefaultUser=*user:group*	[**u**] Set the default userid for mailers to *user:group*. If *group* is omitted and *user* is a user name (as opposed to a numeric user id), the default group listed in the **/etc/passwd** file for that user is used as the default group. Both user and group may be numeric. Mailers without the **S** flag in the mailer definition will run as this user. Defaults to 1:1. The value can also be given as a symbolic user name.

DeliveryMode=*x*

[**d**] Deliver in mode *x*. Legal modes are:

i	Deliver interactively (synchronously)
b	Deliver in background (asynchronously)
q	Just queue the message (deliver during queue run)
d	Defer delivery and all map lookups (deliver during queuerun) (This mode is available beginning with AIX Version 4.2).

Defaults to **b** if no option is specified, **i** if it is specified but given no argument (for example, ‘‘Od’’ is equivalent to ’‘Odi’’). The **-v** command line flag sets this to **i**.

DialDelay=*sleeptime*

[no short name] Dial-on-demand network connections can see timeouts if a connection is opened before the call is set up. If this is set to an interval and a connection times out on the first connection being attempted, **sendmail** will sleep for this amount of time and try again. This should give your system time to establish the connection to your service provider. Units default to seconds, so "DialDelay=5" uses a five second delay. Defaults to zero (no retry).

DontBlameSendmail=*option,option,...*	[no short name] To avoid possible cracking attempts caused by world-writeable and group-writeable files and directories, **sendmail** does paranoid checking when opening most of its support files. If you must use a group-writeable directory (e.g., **/etc**), you must turn off this checking. Turning off checking will make your system more vulnerable to attack. The following arguments are individual options that turn off checking: **Note:** *Safe* is the default option, but it is *not* recommended for use. Safe AssumeSafeChown ClassFileInUnsafeDirPath ErrorHeaderInUnsafeDirPath FileDeliveryToHardLink FileDeliveryToSymLink ForwardFileInUnsafeDirPath ForwardFileInUnsafeDirPathSafe ForwardFileInGroupWritableDirPath GroupWritableAliasFile GroupWritableDirPathSafe GroupWritableForwardFileSafe GroupWritableIncludeFileSafe HelpFileInUnsafeDirPath IncludeFileInUnsafeDirPath IncludeFileInUnsafeDirPathSafe IncludeFileInGroupWritableDirPath LinkedAliasFileInWritableDir LinkedClassFileInWritableDir LinkedForwardFileInWritableDir LinkedIncludeFileInWritableDir LinkedMapInWritableDir LinkedServiceSwitchFileInWritableDir MapInUnsafeDirPath RunProgramInUnsafeDirPath RunWritableProgram WorldWritableAliasFile WriteMapToHardLink WriteMapToSymLink WriteStatsToHardLink WriteStatsToSymLink
DontExpandCnames	[no short name] The standards say that all host addresses used in a mail message must be fully canonical. For example, if your host is named "Cruft.Foo.ORG" and also has an alias of "FTP.Foo.ORG", the former name must be used at all times. This is enforced during host name canonification ($[... $] lookups). If this option is set, the protocols are ignored and the "wrong" thing is done. However, the IETF is moving toward changing this standard, so the behavior may become acceptable. Please note that hosts downstream may still rewrite the address to be the true canonical name.
DontInitGroups	[no short name] If set, **sendmail** will avoid using the initgroups(3) call. If you are running NIS, this causes a sequential scan of the groups.byname map, which can cause your NIS server to be badly overloaded in a large domain. The cost of this is that the only group found for users will be their primary group (the one in the password file), which will make file access permissions somewhat more restrictive. Has no effect on systems that do not have group lists.

DontProbeInterfaces	Normally, when **sendmail** starts, it finds the names of all interfaces that are active on your machine and adds these names to the **$=w** class of known host aliases. This activity can take a long time if you have a large number of virtual interfaces or if your DNS inverse lookups are slow. You can turn off this probing by using this option; however, you must use some other mechanism to include all variant names in the **$=w** class.
DontPruneRoutes	[**R**] Normally, **sendmail** tries to eliminate any unnecessary explicit routes when sending an error message (as discussed in RFC 1123 S 5.2.6). For example, when sending an error message to: `<@known1,@known2,@known3:user@unknown>` **sendmail** will strip off the "@known1,@known2" in order to make the route as direct as possible. However, if the **R** option is set, this will be disabled, and the mail will be sent to the first address in the route, even if later addresses are known. This may be useful if you are behind a firewall.
DoubleBounceAddress=*error-address*	[no short name] If an error occurs when sending an error message, send the error report (termed a "double bounce" because it is an error "bounce" that occurs when trying to send another error "bounce") to the indicated address. If not set, defaults to "postmaster". This option is valid for sendmail version 8.8 and above.
EightBitMode=*action*	[**8**] Set handling of eight-bit data. There are two kinds of eight-bit data: that declared as such using the BODY=8BITMIME ESMTP declaration or the -B8BITMIME command line flag, and undeclared 8-bit data, that is, input that just happens to be eight bits. There are three basic operations that can happen: undeclared 8-bit data can be automatically converted to 8BITMIME, undeclared 8-bit data can be passed as is, without conversion to MIME, and declared 8-bit data can be converted to 7-bits for transmission to a non-8BITMIME mailer. The possible actions are: **s** Reject undeclared 8-bit data (‘‘strict’’) **m** Convert undeclared 8-bit data to MIME (‘‘mime’’) **p** Pass undeclared 8-bit data (‘‘pass’’) In all cases, properly declared 8BITMIME data will be converted to 7BIT as needed.
ErrorHeader=*file-or-message*	[**E**] Prepend error messages with the indicated message. If it begins with a slash, it is assumed to be the pathname of a file containing a message (this is the recommended setting). Otherwise, it is a literal message. The error file might contain the name, e-mail address, and/or phone number of a local postmaster who could provide assistance to end users. If the option is missing or null, or if it names a file which does not exist or which is not readable, no message is printed.

ErrorMode=*x*	[**e**] Dispose of errors using mode *x*. The values for *x* are: **p** Print error messages (default) **q** No messages, just give exit status **m** Mail back errors **w** Write back errors (mail if user not logged in) **e** Mail back errors and give zero exit stat always
FallbackMXhost=*fallbackhost*	[**V**] If specified, the *fallbackhost* acts like a very low priority MX on every host. This is intended to be used by sites with poor network connectivity.
ForkEachJob	[**Y**] If set, deliver each job that is run from the queue in a separate process. Use this option if you are short of memory, since the default tends to consume considerable amounts of memory while the queue is being processed.
ForwardPath=*path*	[**J**] Set the *path* for searching for users' **.forward** files. The default is "$z/.forward". Some sites that use the automounter may prefer to change this to "/var/forward/$u" to search a file with the same name as the user in a system directory. It can also be set to a sequence of paths separated by colons; **sendmail** stops at the first file it can successfully and safely open. For example, "/var/forward/$u:$z/.forward" will search first in **/var/forward/username** and then in **~username/.forward** (but only if the first file does not exist).
HoldExpensive	[**c**] If an outgoing mailer is marked as being expensive, don't connect immediately. This requires that queueing be compiled in, since it will depend on a queue run process to actually send the mail.
HostsFile=*path*	[no short name] The path to the hosts database, normally */etc/hosts*. This option is only consulted when **sendmail** is canonifying addresses, and then only when "files" is in the "hosts" service switch entry. In particular, this file is never used when looking up host addresses; that is, under the control of the system **gethostbyname(3)** routine. This option is valid for sendmail version 8.8 and above.
HostStatusDirectory=*path*	[no short name] The location of the long term host status information. When set, information about the status of hosts (for example, host down or not accepting connections) will be shared between all sendamil processes; normally, this information is only held within a single queue run. This option requires a connection cache of at least 1 to function. If the option begins with a leading '/', it is an absolute pathname; otherwise, it is relative to the mail queue directory. A suggested value for sites desiring persistent host status is *.hoststat* (that is, a subdirectory of the queue directory). This option is valid for sendmail version 8.8 and above.
IgnoreDots	[**i**] Ignore dots in incoming messages. This is always disabled (that is, dots are always accepted) when reading SMTPmail.
LogLevel=*n*	[**L**] Set the default log level to *n*. Defaults to 9.

M*xvalue*	[no long version] Set the macro *x* to *value*. This is intended only for use from the command line. The **-M** flag is preferred.
MatchGECOS	[**G**] Allow fuzzy matching on the GECOS field. If this flag is set, and the usual user name lookups fail (that is, there is no alias with this name and a getpwnam fails), sequentially search the password file for a matching entry in the GECOS field. This also requires that MATCHGECOS be turned on during compilation. This option is not recommended.
MaxDaemonChildren=*N*	[no short name] If set, **sendmail** will refuse connections when it has more than *N* children processing incoming mail. This does not limit the number of outgoing connections. If not set, there is no limit to the number of children; that is, the system load averaging controls this. This option is valid for sendmail version 8.8 and above.
MaxHopCount=*N*	[**h**] The maximum hop count. Messages that have been processed more than *N* times are assumed to be in a loop and are rejected. Defaults to 25.
MaxMessageSize=*N*	[no short name] Specify the maximum message size to be advertised in the ESMTP EHLO response. Messages larger than this will be rejected.
MaxQueueRunSize=*N*	[no short name] The maximum number of jobs that will be processed in a single queue run. If not set, there is no limit on the size. If you have very large queues or a very short queue run interval, this could be unstable. However, since the first *N* jobs in queue directory order are run (rather than the *N* highest priority jobs), this should be set as high as possible to avoid "losing" jobs that happen to fall late in the queue directory.
MaxRecipientsPerMessage=*N*	[no short name] The maximum number of recipients that will be accepted per message in an SMTP transaction. If not set, there is no limit on the number of recipients per envelope. **Note:** Setting this option too low can interfere with sending mail from MUAs that use SMTP for initial submission.
MeToo	[**m**] Send to me too, even if I am in an alias expansion.
MinFreeBlocks=*N*	[**b**] Insist on at least *N* blocks free on the filesystem that holds the queue files before accepting e-mail via SMTP. If there is insufficient space, **sendmail** gives a 452 response to the MAIL command. This invites the sender to try again later.
MinQueueAge=*age*	[no short name] Don’t process any queued jobs that have been in the queue less than the indicated time interval. This is intended to allow you to get responsiveness by processing the queue fairly frequently without thrashing your system by trying jobs too often. The default units are minutes.
MustQuoteChars=*s*	[no short name] Sets the list of characters that must be quoted if used in a full name; that is, in the phrase part of a ‘‘phrase <address>’’ syntax. The default is ‘‘.’’. The characters ‘‘@,;:\()[]’’ are always added to this list. This option is valid for sendmail version 8.8 and above.

NameServOpt

Set this option to use MB, MG, and MR resource records in the name server.

MB	Uses mailbox (MB) records to resolve recipient user names.
MG	Uses mail group (MG) records to resolve recipient user names.
MR	Uses mail rename (MR) records to resolve recipient user names.

NoRecipientAction

[no short name] The action to take when you receive a message that has no valid recipient headers (To:, Cc:, Bcc:). It can be:

None	To pass the message on unmodified, which violates the protocol
Add-To	To add a To: header with any recipients it can find in the envelope (which might expose Bcc: recipients)
Add-Apparently-To	To add an Apparently-To: header (this is only for back compatibility and is officially deprecated)
Add-To-Undisclosed	To add a header "To: undisclosed-recipients:;" to make the header legal without disclosing anything
Add-Bcc	To add an empty Bcc: header.

OldStyleHeaders

[**o**] Assume that the headers may be in old format, that is, spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names. Defaults to off.

OperatorChars=*charlist*

[**$o** macro] The list of characters that are considered to be "operators;" that is, characters that delimit tokens. All operator characters are tokens by themselves; sequences of non-operator characters are also tokens. White space characters separate tokens but are not tokens themselves. For example, "AAA.BBB" has three tokens, but "AAA BBB" has two. If not set, OperatorChars defaults to ".:@[]"; additionally, the characters "()<>,;" are always operators.

PostmasterCopy=*postmaster*

[**P**] If set, copies of error messages will be sent to the named *postmaster*. Only the header of the failed message is sent. Since most errors are user problems, this is probably not a good idea on large sites, and arguably contains privacy violations. Defaults to no postmaster copies.

PrivacyOptions=*opt,opt,...*	[**p**] Set the privacy options. "Privacy" is really a misnomer; many of these are just a way of insisting on stricter adherence to the SMTP protocol. The options can be selected from:

public	Allow open access
needmailhelo	Insist on HELO or EHLO command before MAIL
needexpnhelo	Insist on HELO or EHLO command before EXPN
noexpn	Disallow EXPN entirely
needvrfyhelo	Insist on HELO or EHLO command before VRFY
novrfy	Disallow VRFY entirely
noetrn	Disalllow ETRN entirely
noverb	Disallow VERB entirely
restrictmailq	Restrict **mailq** command
restrictqrun	Restrict **-q** command line flag
noreceipts	Do not return success DSNs
goaway	Disallow essentially all SMTP status queries
authwarnings	Put X-Authentication-Warning: headers in messages

The "goaway" pseudo-flag sets all flags except **restrictmailq** and **restrictqrun**. If mailq is restricted, only people in the same group as the queue directory can print the queue. If queue runs are restricted, only root and the owner of the queue directory can run the queue. Authentication Warnings add warnings about various conditions that may indicate attempts to violate the mail system security, such as using an nonstandard queue directory.

QueueDirectory=*dir*	[**Q**] Use the named *dir* as the queue directory.
QueueFactor=*factor*	[**q**] Use *factor* as the multiplier in the map function to decide when to just queue up jobs rather than run them. This value is divided by the difference between the current load average and the load average limit (QueueLA option) to determine the maximum message priority that will be sent. Defaults to 600000.
QueueLA=*LA*	[**x**] When the system load average exceeds *LA*, just queue messages (do not try to send them). Defaults to 8.
QueueSortOrder=*algorithm*	[no short name] Sets the *algorithm* used for sorting the queue. Only the first character of the value is used. Legal values are "host" (to order by the name of the first host name of the first recipient) and "priority" (to order strictly by message priority). Host ordering makes better use of the connection cache, but may tend to process low-priority messages that go to a single host over high-priority messages that go to several hosts; it probably should not be used on slow network links. Priority ordering is the default.
QueueTimeout=*timeout*	[**T**] A synonym for "Timeout.queuereturn". This option is valid for sendmail version 8.8 and above.

RecipientFactor=*fact* — [**y**] The indicated factor is added to the priority (thus lowering the priority of the job) for each recipient. That is, this value penalizes jobs with large numbers of recipients. Defaults to 30000.

RefuseLA=*LA* — [**X**] When the system load average exceeds *LA*, refuse incoming SMTP connections. Defaults to 12.

ResolverOptions=*options* — [**I**] Set resolver *options*. Values can be set using **+**flag and cleared using **-** flag; the flags can be "debug", "aaonly", "usevc", "primary", "igntc", "recurse", "defnames", "stayopen",or "dnsrch". The string "HasWildcardMX " (without a **+** or **-**) can be specified to turn off matching against MX records when doing name canonifications.

You must be using the name server when defining this option.

RetryFactor=*fact* — [**Z**] The factor is added to the priority every time a job is processed. Thus, each time a job is processed, its priority will be decreased by the indicated value. In most environments this should be positive, since hosts that are down can be down for a long time. Defaults to 90000.

RunAsUser=*user* — [no short name] The *user* parameter may be a user name (looked up in */etc/passwd*) or a numeric user id; either form can have ":group" attached (where group can be numeric or symbolic). If set to a non-zero (non-root) value, **sendmail** will change to this user id shortly after startup[20]. This avoids a certain class of security problems. However, this means that all ".forward" and ":include:" files must be readable by the indicated user, and on systems that don’t support the saved uid bit properly, all files to be written must be writable by user and all programs will be executed by user. It is also incompatible with the **SafeFileEnvironment** option. In other words, it may not actually add much to security on an average system, and may, in fact, detract from security (because other file permissions must be loosened). However, it should be useful on firewalls and other places where users don’t have accounts and the aliases file is well constrained. This option is valid for sendmail version 8.8 and above.

SafeFileEnvironment=*dir* — [no short name] If this option is set, **sendmail** will do a chroot(2) call into the indicated directory before doing any file writes. If the file name specified by the user begins with dir, that partial path name will be stripped off before writing, so (for example) if the **SafeFileEnvironment** variable is set to "/safe", then aliases of "/safe/logs/file" and "/logs/file" actually indicate the same file. Additionally, if this option is set, **sendmail** refuses to deliver to symbolic links.

SaveFromLine — [**f**] Save UNIX-style "From" lines at the front of headers. Normally they are assumed redundant and discarded.

SendMIMEErrors — [**j**] If set, send error messages in MIME format (see RFC1521 and RFC1344 for details).

SevenBitInput — [**7**] Strip input to seven bits for compatibility with old systems. This should not be necessary.

SingleLineFromHeader	[no short name] If set, From: lines that have embedded newlines are unwrapped onto one line. This is to get around a botch in Lotus Notes that apparently cannot understand legally wrapped RFC822 headers. This option is valid for sendmail version 8.8 and above.
SingleThreadDelivery	[no short name] If set, a client machine will never try to open two SMTP connections to a single server machine at the same time, even in different processes. That is, if another sendmail is already talking to some host, a new sendmail will not open another connection. This property is of mixed value; although this reduces the load on the other machine, it can cause mail to be delayed (for example, if one sendmail is delivering a huge message, other sendmails won't be able to send even small messages). Also, it requires another file descriptor (for the lock file) per connection, so you may have to reduce the **ConnectionCacheSize** option to avoid running out of per-process file descriptors. Requires the **HostStatusDirectory** option. This option is valid for sendmail version 8.8 and above.
SmtpGreetingMessage=*message*	[**$e** macro] The *message* printed when the SMTP server starts up. Defaults to "$j **Sendmail** $v ready at $b".
StatusFile=*file*	[**S**] Log summary statistics in the named *file*. If not set, no summary statistics are saved. This file does not grow in size. It can be printed using the **mailstats** program.
SuperSafe	[**s**] Be super-safe when running things. That is, always instantiate the queue file, even if you are going to attempt immediate delivery. **sendmail** always instantiates the queue file before returning control to the client under any circumstances. It is recommended that this always be set.
TempFileMode=*mode*	[**F**] The file *mode* for queue files. It is interpreted in octal by default. Defaults to 0600.

Timeout.*type*=*timeout*

[**r**; subsumes old **T** option as well] Set timeout values. The actual timeout is indicated by the *type*. The recognized timeouts and their default values, and their minimum values specified in RFC 1123 section 5.3.2 are:

initial	Wait for initial greeting message [5m, 5m]
helo	Reply to HELO or EHLO command [5m, none]
mail	Reply to MAIL command [10m, 5m]
rcpt	Reply to RCPT command [1h, 5m]
datainit	Reply to DATA command [5m, 2m]
datablock	Data block read [1h, 3m]
datafinal	Reply to final "." in data [1h, 10m]
rset	Reply to RSET command [5m, none]
quit	Reply to QUIT command [2m, none]
misc	Reply to NOOP and VERB commands [2m, none]
ident	IDENT protocol timeout [30s, none]
fileopen-	Timeout on opening **.forward** and **.include**: files [60s, none]
command-	Command read [1h, 5m]
queuereturn-	How long until a message is returned [5d, 5d]
queuewarn-	How long until a warning is sent [none, none]
hoststatus-	How long until host status is "stale" [30m, none]

All but those marked with a (-) apply to client SMTP. If the message is submitted using the NOTIFY SMTP extension, warning messages will only be sent if NOTIFY=DELAY is specified. The **queuereturn** and **queuewarn** timeouts can be further qualified with a tag based on the Precedence: field in the message; they must be one of the following:

urgent	Indicating a positive non-zero precedence
normal	Indicating a zero precedence
nonurgent	Indicating negative precedences.

For example, setting "Time-out.queuewarn.urgent=1h" sets the warning timeout for urgent messages only to one hour. The default if no precedence is indicated is to set the timeout for all precedences.

TimeZoneSpec=*tzinfo*	[**t**] Set the local time zone info to *tzinfo*; for example, "PST8PDT". Actually, if this is not set, the TZ environment variable is cleared (so the system default is used). If set but null, the user's TZ variable is used, and if set and non-null, the TZ variable is set to this value.
TryNullMXList	[**w**] If this system is the "best" (that is, lowest preference) MX for a given host, its configuration rules should normally detect this situation and treat that condition specially by forwarding the mail to a UUCP feed, treating it as local. However, in some cases (such as Internet firewalls), you may want to try to connect directly to that host as though it had no MX records at all. Setting this option causes **sendmail** to try this. Errors in your configuration are likely to be diagnosed as "host unknown" or "message timed out" instead of something more meaningful. This option is not recommended.
UnixFromLine=*fromline*	[**$l** macro] Defines the format used when **sendmail** must add a UNIX-style From_ line (that is, a line beginning "From<space>user"). Defaults to "From $g $d". Do not change this unless your system uses a different UNIX mailbox format.
UnsafeGroupWrites	[no short name] If set, :include: and .forward files that are group writable are considered "unsafe", that is, they cannot reference programs or write directly to files. World writable :include: and .forward files are always unsafe. This option is valid for sendmail version 8.8 and above.
UseErrorsTo	[**l**] If there is an "Errors-To:" header, send error messages to the addresses listed there. They normally go to the envelope sender. Use of this option causes **sendmail** to violate RFC 1123. This option is not recommended.
UserSubmission	[no short name] This is an initial submission directly from a Mail User Agent. This can be set in the configuration file if you have MUAs that do not pass the **-U** flag or use the XUSR ESMTP extension, but some relayed mail may get inappropriately rewritten if you do.
Verbose	[**v**] Run in verbose mode. If this is set, **sendmail** adjusts options HoldExpensive (old **c**) and DeliveryMode (old **d**) so that all mail is delivered completely in a single job so that you can see the entire delivery process. Option Verbose should never be set in the configuration file; it is intended for command line use only.

All options can be specified on the command line using the **-O** or **-o** flag, but most will cause **sendmail** to relinquish its setuid permissions. The options that will not cause this are MinFreeBlocks [b], DeliveryMode [d], ErrorMode [e], IgnoreDots [i], LogLevel [L], MeToo [m], OldStyleHeaders [o], PrivacyOptions [p], Timeouts [r], SuperSafe [s], Verbose [v], CheckpointInterval [C], and SevenBitInput [7]. Also, M (define macro) when defining the r or s macros is also considered "safe."

Note: The ServicesSwitchFile option is not supported. AIX **sendmail** uses AIX name resolution, which means that it uses either **/etc/netsvc.conf**, the NSORDER environment variable, or the default name resolution (DNS). If you do not want **sendmail** to use the DNS, you must specify the type of name resolution to use in either **/etc/netsvc.conf** or the NSORDER environment variable (for example, `NSRODER=local`). For more information, see "Name Resolution" in the TCP/IP chapter of *AIX Version 4.3 System Management Guide: Communications and Networks*

sendmail.cf File Options (for AIX Version 3.2.5 or AIX Version 4.1)

Set configuration options for the **sendmail** command by using a control line in the configuration file. The options are the same as those specified with the **sendmail** command and the **-o** flag. Name an option with a single character; for example:

```
OOption[Value]
```

Where *Option* is a single-character name for the option being set. *Value* is either a string, an integer, a time interval, or a Boolean option. Values for a Boolean option are **t**, **T**, **y**, **Y** for a true value and **f** or **F** for a false value. If you do not specify a value for a Boolean option, the default is true.

The options supported are as follows:

A*File*
: Uses the *File* specified as the alias file.

b*CodeSet*
: Specifies the National Language Support code set of the network in Japanese environments. The **sendmail** command converts mail to or from the code set of the network from or to the code set of the locale. For NLS code set values see, "Understanding ISO Code Sets" or "Understanding IBM PC Code Sets" in *AIX Kernel Extensions and Device Support Programming Concepts*.

B*Character*
: Sets the blank substitution *Character*. In addresses, the **sendmail** command replaces spaces without quotes with the specified *Character*. The supplied configuration file uses a . (period) for the value of the *Character* variable. This option prohibits the **sendmail** command from receiving messages through SMTP. The **sendmail** command cannot recognize the end of the message, a . (period), when receiving messages through SMTP.

c
: Causes the **sendmail** command to queue messages without sending them if an outgoing mailer is marked as expensive to use. The queue can be run when costs are lower or when the queue is large enough to send the message efficiently.

d
: The **sendmail** command operates in several delivery modes. The default configuration file sets the delivery mode to b (the default value). However, you can change the **Od** delivery mode with the **Od***Value* option in the configuration file. This mode specifies how promptly mail is delivered. *Value* can be:

  i Delivers interactively

  b Delivers in background (the default)

  q Queues the message and delivers during queue run.

e	Dispose of errors using mode *x*. The values for *x* are: **p** Print error messages (default) **q** No messages, just give exit status **m** Mail back errors **w** Write back errors (mail if user not logged in) **e** Mail back errors and give zero exit stat always
E*TimeValue*	To comply with RFC 1123, which specifies per-command time-outs for the SMTP protocol, uncomment this line in the **/etc/sendmail.cf** configuration file, where *TimeValue* is the length of time the local SMTP protocol waits, after sending the **rcpt** command, for a reply from the remote SMTP protocol.
f	Saves `From` lines at the front of messages. These lines are normally discarded.
F*Value*	Sets the file mode for temporary files created by **sendmail** in the queue directory. *Value* is an octal file mode value. The default temporary file mode is 0660.
g*Number*	Sets the default group ID to the value specified by the *Number* variable.
G*TimeValue*	To comply with RFC 1123, which specifies per-command time-outs for the SMTP protocol, uncomment this line in the **/etc/sendmail.cf** configuration file, where *TimeValue* is the length of time the SMTP protocol waits for completion of sending data.
h	The **sendmail Oh** option limits the number of alternate addresses tried for multihomed hosts. Without the option, only the first address is tried. The format for this option is: `Oh`*`Number`* The *Number* variable specifies the number of alternate addresses to be tried.
i	Does not interpret a . (period) on a line by itself as a message terminator. This option prohibits the **sendmail** command from receiving messages through SMTP. The **sendmail** command cannot recognize the end of the message, a . (period), when receiving messages through SMTP.
I	Treats a failure to connect to the name server as a temporary error. The message is queued and delivery can be retried later.
J	Allows spaces as well as tabs to separate the left-hand side and right-hand side of rewrite rules. This option allows rewrite rules to be modified using an editor that replaces tabs with spaces.
k	Prevents character-set conversions for outgoing mail messages. Any National Language Support (NLS) extended characters are sent to other systems intact.

K*[Value*] Sets the types of name server resource records that the **sendmail** command uses to resolve recipient addresses. If using more than one **K**Value option, separate them with a space. *Value* can be:

- **MB** Uses mailbox (MB) records to resolve recipient user names.
- **MG** Uses mail group (MG) records to resolve recipient user names.
- **MR** Uses mail rename (MR) records to resolve recipient user names.
- **MX** Uses mail exchanger (MX) records to resolve recipient users.
- **ANY** Query for ANY records.
- **ALL** Uses all of the above.

l*File* Sets the National Language Support (NLS) configuration file to *File*.

m Sends messages to the sender if the sender appears in an alias expansion of a recipient address. The default action removes the sender's address from recipient alias expansions.

M*Macro Value* Defines the *Macro* variable as the *Value* specified. This option is normally used from the **sendmail** command line only.

n Validates the right-hand side of alias definitions when performing the **newaliases** function.

o Indicates that this message can have old-style headers. Without this option, the message has new style headers (commas instead of spaces between addresses). If this option is set, an adaptive algorithm correctly determines the header format in most cases.

OCodeSet Specifies the National Language Support code set of the local site in Japanese environments. The **sendmail** command converts mail to or from the code set of the locale to or from the code set of the network. For NLS code set values see, "Understanding ISO Code Sets" or "Understanding IBM PC Code Sets" in *AIX Kernel Extensions and Device Support Programming Concepts*.

p*MapName* Sets the map name of NIS aliases when using NIS. If mail aliases are not found in the local **/etc/aliases** file, a search begins in a domainwide NIS database, usually located on a central machine.

P*Address* Identifies the *Address* to receive a copy of all returned mail.

q*Value* Use *value* as the multiplier in the map function to decide when to just queue up jobs rather than run them. This *value* is divided by the difference between the current load average and the load average limit (**x** option) to determine the maximum message priority that will be sent. Defaults to 600000.

Q*Directory* Sets the directory for queuing messages. A directory is created if one does not exist.

R*TimeValue*	To comply with RFC 1123, which specifies per-command time-outs for the SMTP protocol, uncomment this line in the **/etc/sendmail.cf** configuration file, where *TimeValue* is the length of time the local SMTP protocol waits, after sending the **data** command, for a reply from the remote SMTP protocol.
s	Enqueues messages before delivery, even during immediate delivery mode.
S*File*	Specifies the path name of the *File* where the **sendmail** command stores data about delivered and received messages. Statistics are only collected if the file exists. This file must be created by the user.
u*Number*	Sets the default user ID to the value specified by the *Number* variable.
U*TimeValue*	To comply with RFC 1123, which specifies per-command time-outs for the SMTP protocol, uncomment this line in the **/etc/sendmail.cf** configuration file, where *TimeValue* is the length of time the local SMTP protocol waits for the 220 greeting message from the remote SMTP protocol.
v	Runs in verbose mode.
V*TimeValue*	To comply with RFC 1123, which specifies per-command time-outs for the SMTP protocol, uncomment this line in the **/etc/sendmail.cf** configuration file, where *TimeValue* is the length of time the local SMTP protocol waits, after sending the **mail** command, for a reply from the remote SMTP protocol.
w	Causes all incoming mail that has 8-bit characters to be treated as ISO-8859/1 mail. These characters are converted to the equivalent National Language Support (NLS) extended characters.
W*TimeValue*	To comply with RFC 1123, which specifies per-command time-outs for the SMTP protocol, uncomment this line in the **/etc/sendmail.cf** configuration file, where *TimeValue* is the length of time the local SMTP protocol waits, after sending the mail termination **.** (period) command, for a reply from the remote SMTP protocol.
x=*Value*	When the system load average exceeds *value*, just queue messages (do not try to send them). Defaults to 8.
X=*Value*	When the system load average exceeds *value*, refuse incoming SMTP connections. Defaults to 12.
y=*Value*	The indicated *value* is added to the priority (thus lowering the priority of the job) for each recipient. That is, this *value* penalizes jobs with large numbers of recipients. Defaults to 30000.
Y	Delivers each message in the mail queue from a separate process. This option is not required. If used, the option increases overhead in the workstation environment. In addition, unavailable destination hosts may be retried during a queue run.
+	Turns on secure SMTP. When enabled, this option disables the VRFY and EXPN commands. These commands are required and do run, but they echo their argument back to the user rather than expanding the argument to indicate whether it is valid or invalid.

-	Turns on SMTP security logging. When enabled, any use of the VRFY and EXPN commands is logged, even if the commands are disabled by the **+** option. Any invalid user given to the RCPT command is also logged. The log message is sent to **syslogd** as a **mail.warning** message. The message includes the date, time, user's hostname, command, and argument given to SMTP.

Implementation Specifics

This **sendmail.cf** file is part of Base Operating System (BOS) Runtime.

Files

/etc/sendmail.cf	Specifies the path of the **sendmail.cf** file.
/etc/passwd	Contains basic user attributes.
/etc/aliases	Contains alias definitions for the **sendmail** command.

Related Information

The **sendmail** command.

setup.csh File

Purpose

Sets the C-shell environment variables needed to build an InfoCrafter database.

Description

The **setup.csh** file defines C-shell environment variables necessary to build an InfoCrafter database from the command line. The **setup.csh** file contains the definition of the **TOOLSDIR** and **TOPLEVEL_BUILDDIR** variables; if there are relative path names of source files in your input list, it also sets the **TOPLEVEL_SOURCEDIR** variable. The **TOOLSDIR** variable is added to your path environment variable so you can use the **icft** command without specifying the full path name.

The default value for the **TOOLSDIR** environment variable is **/usr/lpp/icraft/bin**. The **TOPLEVEL_SOURCEDIR** and **TOPLEVEL_BUILDDIR** variables have no default values.

You must copy the **setup.csh** file from **/usr/lpp/icraft/bin** to another location (such as your home directory) and edit it to define the variables. Then, use the **source setup.csh** command to assign the new definitions to the variables.

Examples

A sample **setup.csh** file appears as follows:

```
setenv          TOPLEVEL_SOURCEDIR      $HOME/desktop
setenv          TOOLSDIR                /usr/lpp/icraft/bin
setenv          TOPLEVEL_BUILDDIR       $TOOLSDIR/master
```

To set the C-shell environment variables, enter the following:

```
source setup.csh
```

The following message is displayed:

```
setup.csh:      assigning environment variables
                for InfoCrafter. . .
```

Implementation Specifics

This file is part of the InfoCrafter product.

Files

/usr/lpp/icraft/bin/setup.csh	Contains the definitions of C-shell environment variables.

Related Information

setup.sh File

Purpose

Defines the Bourne or Korn shell environment variables needed to build an InfoCrafter database.

Description

The **setup.sh** file defines Bourne or Korn shell environment variables necessary to build an InfoCrafter database from the command line using the **icft** command. The **setup.sh** file sets the **TOOLSDIR** and **TOPLEVEL_BUILDDIR** variables. If there are relative path names of source files in your input list, it also sets the **TOPLEVEL_SOURCEDIR** variable. The **TOOLSDIR** variable is added to your path environment variable so you can enter the **icft** command without specifying the full path name.

Default value for the **TOOLSDIR** environment path variable is **/usr/lpp/icraft/bin**. **TOPLEVEL_SOURCEDIR** and **TOPLEVEL_BUILDDIR** have no default values.

You must copy the **setup.sh** file from **/usr/lpp/icraft/bin** to another location (such as your home directory) and edit it to define the variables. Then, use the **. setup.sh** command to set the variables to the defined values.

Examples

A sample **setup.sh** file appears as follows:

```
TOPLEVEL_SOURCEDIR =             $HOME/desktop
TOOLSDIR =                       /usr/lpp/icraft/bin
TOPLEVEL_BUILDDIR =              $TOOLSDIR/master
```

To set Bourne or Korn shell environment variables, enter the following:

```
. setup.sh
```

The following message is given:

```
setup.sh: assigning environment variables for InfoCrafter
```

Implementation Specifics

This file is part of the InfoCrafter product.

Files

/usr/lpp/icraft/bin/setup.sh	Contains definitions for Bourne and Korn shell environment variables.

Related Information

smi.my File

Purpose

Provides sample SMI input to the **mosy** command.

Description

The **/usr/samples/snmpd/smi.my** file is a sample input file to the **mosy** command, which creates an objects definition file for use by the **snmpinfo** command. The **mosy** compiler requires its input file to contain the ASN.1 definitions described in the Structure and Identification of Management Information (SMI) RFC 1155 and the Management Information Base (MIB) RFC 1213. The **smi.my** file contains the syntax descriptions from the SMI RFC 1155.

The **smi.my** file begins with a definition of the SNMP subtree of the MIB as assigned by the Internet Activities Board (IAB). It then contains the syntax definitions defined in RFC 1155.

Comments are specified by **- -** (two dashes). A comment can begin at any location and extends to the end of the line.

The **smi.my** file was created by extracting the definitions from Chapter 6 of RFC 1155. This file is shipped as **/usr/samples/snmpd/smi.my**.

Implementation Specifics

This file is part of Simple Network Management Protocol Agent Applications in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/usr/samples/snmpd/mibII.my	Contains the ASN.1 definitions for the MIB II variables defined in RFC 1213.
/etc/mib.defs	Defines the Management Information Base (MIB) variables the **snmpd** agent should recognize and handle. This file is in the format that the **snmpinfo** command requires.

Related Information

The **mosy** command, **snmpinfo** command.

The **mibII.my** file.

RFC 1155, RFC 1213.

smitacl.group File

Purpose

Contains the group access control list (ACL) definitions for the System Management Interface Tool (SMIT). This system file only applies to AIX Versions 4.2.1 and later.

Description

The **/etc/security/smitacl.group** file contains the group ACL definitions for SMIT. This is an ASCII file that contains a stanza for each system group. Each stanza is identified by a group name followed by a : (colon) and contains attributes in the form *Attribute=Value*. Each attribute pair ends with a newline character as does each stanza.

The file supports a default stanza. If an attribute is not defined, either the default stanza or the default value for the attribute is used.

A stanza contains the following attribute:

screens Describes the list of SMIT screens for this group. (It is of the type **SEC_LIST**.) Examples include:

```
screens = *                   # Permit all screen access.
screens = !*                  # Deny all screen access.
screens =                     # No specific screens.
screens = user,group,!tcpip # Allow user & group
                              # screens, but not
                              # tcpip screen
```

For a typical stanza, see the "Examples" section.

Security

Access Control: This file grants read and write access to the root user, and read access to members of the security group.

Examples

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

```
group:
        screens = *
```

Implementation Specifics

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

Files

/etc/security/roles	Contains the list of valid roles.
/etc/security/user.roles	Contains the list of roles for each user.
/etc/security/smitacl.group	Contains the group ACL definitions.
/etc/security/smitacl.user	Contains the user ACL definitions.

Related Information

smitacl.user File

Purpose

Contains the user access control list (ACL) definitions for the System Manamgement Interface Tool (SMIT). This system file only applies to AIX Versions 4.2.1 and later.

Description

The **/etc/security/smitacl.user** file contains the ACL definitions for SMIT. This is an ASCII file that contains a stanza for each system user. Each stanza is identified by a user name followed by a : (colon) and contains attributes in the form *Attribute=Value*. Each attribute pair ends with a newline character as does each stanza.

The file supports a default stanza. If an attribute is not defined, either the default stanza or the default value for the attribute is used.

A stanza contains the following attributes:

screens
Describes the list of SMIT screens for the user. (It is of the type **SEC_LIST**.) Examples include:

```
screens = *                       # Permit all screen access.
screens = !*                      # Deny all screen access.
screens =                         # No specific screens.
screens = user,group,!tcpip       # Allow user & group
                                  # screens, but not
                                  # tcpip screen
```

funcmode
Describes if the role database and/or SMIT ACL database should be used to determine accessibility. It also describes how to combine the **screens** data from the two databases. (It is of the type **SEC_CHAR**.) Examples include:

```
funcmode = roles+acl   # Use both roles and SMIT ACL # databases.
funcmode = roles       # Use only the roles database.
funcmode = acl         # Use only the SMIT ACL # database.
```

The defined values for **funcmode** are:

roles
Only the screen values from the roles database are used.

acl
Only the screen values from the SMIT ACL database are used.

roles+acl
The screen values from both the roles and the SMIT ACL databases are used.

For a typical stanza, see the "Examples" section .

Security

Access Control: This file grants read and write access to the root user, and read access to members of the security group.

Examples

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

```
username:
        screens = *
        funcmode = roles+acl
```

Implementation Specifics

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

Files

/etc/security/roles	Contains the list of valid roles.
/etc/security/user.roles	Contains the list of roles for each user.
/etc/security/smitacl.group	Contains the group ACL definitions.
/etc/security/smitacl.user	Contains the user ACL definitions.

Related Information

snmpd.conf File

Purpose

Defines a sample configuration file for the **snmpd** agent.

Description

The **snmpd.conf** file provides the configuration information for the **snmpd** agent. This file can be changed while the **snmpd** agent is running. If the **refresh** or **kill -1** command is issued, the **snmpd** agent will reread this configuration file. The **snmpd** agent must be under System Resource Control (SRC) for the **refresh** command to force the reread.

This configuration file contains:

- Entries for Community names
- Access privileges and view definitions for incoming Simple Network Management Protocol (SNMP) request packets
- Entries for host destinations for trap notification
- Entries for log file characteristics
- Entries for snmpd-specific parameters
- Entries for SNMP Multiplexing Protocol (SMUX) association configurations
- Entries for the **sysLocation** and **sysContact** variables .

The **snmpd.conf** file must be owned by the root user. If the **snmpd.conf** file is not owned by root, or if the **snmpd** daemon cannot open the configuration file, the **snmpd** daemon issues a **FATAL** message to the logfile if logging is enabled and **snmpd** terminates.

Certain rules apply for specifying particular parameters in entries in the **snmpd.conf** configuration file. Some entries require the specification of object identifiers or object names or both. The following rules apply:

1. An object identifier is specified in dotted numeric notation and must consist of at least three elements. The maximum number of elements in the object identifier is 50. Elements are separated by a . (dot). The first element must be a single digit in the range of 0 to 2. The second element must be an integer in the range of 1 to 40. The third and subsequent elements must be integers in the range of 1 to the size of an unsigned integer.

2. An object name consists of a textual name with an optional numeric instance. The object name must be known to the snmpd agent. Object names typically are names of nodes in the Management Information Base (MIB) tree. If the root of the MIB tree, `iso`, is specified as an object name, the numeric instance is absolutely required. A . (dot) separates the textual name from the numeric instance.

Community Entry

The community entry specifies the communities, associated access privileges and MIB views the **snmpd** agent allows. See example 1 for a sample entry. A community entry must be in the following format:

```
community CommunityName IPAddress NetMask Permissions ViewName
```

The following definitions apply to the variables in a `community` entry:

CommunityName	The community name.
IPAddress	The host name or IP address in dotted-decimal format for the specified community name.
NetMask	A network mask in dotted-decimal format for the specified hostname or IP address.
Permissions	Specifies one of: • readOnly • writeOnly • readWrite • none. The *Permissions* string is case-insensitive.
ViewName	A unique object identifier in dotted numeric notation that is associated with a portion of the MIB tree to which the specified community name allows access. The *ViewName* value is the same as that specified in the view entry.

The minimum specification required for a `community` entry is:

```
community CommunityName
```

The default values for this minimum community entry are:

IPAddress	`0.0.0.0`
NetMask	`0.0.0.0`
Permissions	`readOnly`
View	`iso.3`

Fields to the right of the minimum entry are optional, with the limitation that no fields to the left of a specified field are omitted. Any information to the right of the *ViewName* variable is ignored. If an *IPAddress* of 0.0.0.0 is specified, the default *NetMask* is 0.0.0.0. If an *IPAddress* other than 0.0.0.0 is specified, the default *NetMask* is 255.255.255.255.

The *Permissions* default is readOnly. If the *ViewName* is not specified, the view for this community defaults to ISO, the entire MIB tree. For example:

```
community    public     192.100.154.1
```

is a valid entry with the default values:

NetMask	`255.255.255.255`
Permissions	`readOnly`
View	`iso.3`

The following entry is not valid because the required *NetMask* variable to the left of the *Permissions* variable is not specified:

```
community  public   192.100.154.1
readWrite
```

In this case, the value in the *Permissions* variable is accepted as the *NetMask* value. Since the value in the *Permissions* variable is not in the format required for the *NetMask* variable, an error will occur. The **snmpd** agent logs an EXCEPTIONS message if logging is enabled. In the case of an invalid community entry, the **snmpd** agent ignores the entry.

View Entry

The view entry specifies the MIB subtrees to which a particular community has access. See example 3 for a sample entry. A view entry must be in the following format:

```
view   ViewName       MibSubtree...
```

The following definitions apply to the variables in the `view` entry:

ViewName
: Specifies a unique object identifier in dotted-numeric notation that is associated with a portion of the MIB tree. This *ViewName* value is the same as that in the community entry and must be formatted as described there.

MibSubtree
: A list of MIB subtrees, or MIB groups, specified as either an object name or an object identifier, that is associated with the *ViewName* variable. If the *MIBSubtree* list is not specified, the view defaults to iso, the entire MIB tree.

Together, the view entry and its associated community entry define an access privilege or MIB view allowed by the **snmpd** agent.

In the case of an invalid view entry, the **snmpd** agent logs an EXCEPTIONS message, if logging is enabled, and ignores the view entry.

If a *ViewName* is specified in the community entry, but there is no view entry to describe that *ViewName*, **snmpd** logs an EXCEPTIONS message stating that there is no such view for the community. The **snmpd** agent will allow no access for that community and view association.

Trap Entry

The trap entry specifies the hosts the **snmpd** agent notifies in the event a trap is generated. See Example 2 for a sample entry. A trap entry must be in the following format:

```
trap  CommunityName  IPAddress ViewName TrapMask
```

In this format, the variable definitions are as follows:

CommunityName The community name to be encoded in the SNMP trap packet.

IPAddress The host name or IP Address in dotted-decimal format for the specified *CommunityName*.

ViewName A unique object identifier in dotted numeric notation. *ViewName* is not implemented in Version 3.2. The **snmpd** agent only checks the *ViewName* to verify that the format is valid and that there are no duplicate *ViewName* variables specified.

TrapMask The trap mask in hexadecimal format. The bits from left to right stand for *coldStart* trap, *warmStart* trap, *linkDown* trap, *linkUp* trap, *authenticationFailure* trap, *egpNeighborLoss* trap, and *enterpriseSpecific* trap. The rightmost bit does not have any meaning. A value of 1 will enable the corresponding trap to be sent. Otherwise, the trap is blocked.

For example:

hexadecimal	**bits**	**meaning**
fe	1111 1110	block no traps
7e	0111 1110	block *coldStart* trap
be	1011 1110	block *warmStart* trap
3e	0011 1110	block *coldStart* trap and *warmStart* trap

The minimum specification required for a trap entry is:

```
trap   CommunityName   IPAddress
```

The default value of *TrapMask* for this minimum trap entry is `fe`. There is no trap blocked for this case.

Fields to the right of the minimum entry are optional, with the limitation that no fields to the left of a specified field are omitted. There should be no information to the right of the *TrapMask* variable.

In the case of an invalid trap entry, the **snmpd** agent places an EXCEPTIONS message in the log file if logging is enabled and ignores the trap entry.

It is assumed that all hosts listed in the trap entries are listening on well-known UDP port 162 for SNMP traps. Because community views for traps are not supported, the **snmpd** agent will send trap messages for all traps generated as indicated by the *TrapMask* variable to the hosts listed in the trap entries. If no trap entry appears in the **snmpd.conf** file, the **snmpd** agent will not send out trap messages upon the generation of a trap.

Logging Entry

The logging entry specifies the characteristics for the **snmpd** agent logging activities if logging is not directed from the **snmpd** command with the **-f** option. See example 4 for a sample entry. A logging entry must be in the following format:

```
logging   FileName      Enablement
logging   size=Limit    level=DebugLevel
```

The following definitions apply to the fields in the `logging` entries:

FileName — Specifies the complete path and file name of the log file.

Limit — Specifies the maximum size in bytes of the specified log file. If the limit is specified as 0, the file size is unlimited.

DebugLevel — Specifies the level of logging, which can be one of the following:

- 0 — All NOTICES, EXCEPTIONS, and FATAL messages
- 1 — Level 0 plus DEBUG messages
- 2 — Level 1 plus a hexadecimal dump of incoming and outgoing packets
- 3 — Level 2 plus an English version of the request and response packets

Enablement — Specifies whether logging is active. The following options are available:

- **enabled** — Turns logging on.
- **disabled** — Turns logging off.

There is no default log file. The *Enablement* default is disabled. The log file size *Limit* default is 0, which means unlimited. The *DebugLevel* default is 0 if the **snmpd** command is invoked without the **-d** option. If the **-d** option is specified, the default *DebugLevel* is the value specified by the **-d** option on the **snmpd** command line.

The `size=` and `level=` entries are absolutely required if a size or debug level are specified. There can be no spaces around the `=` (equal sign).

There are no restrictions regarding the order in which the variables are entered in the logging entries. A logging entry can contain single or multiple variables.

If the value for the `size=` field or *DebugLevel* variable cannot be converted into an integer, the default size and debug level are used. Because the **snmpd** command sets the log file configuration parameters immediately upon reading them, the parameters in the logging entry are not necessarily ignored if the **snmpd** command determines there is an invalid field in that entry. For example, in the following invalid logging entry:

```
logging   size=100000   garbagestuff   enabled
```

The **snmpd** command will set the size parameter, but will discard all information from the field value of *garbagestuff* to the end of the line. In addition, an EXCEPTIONS message will be logged if logging is enabled.

snmpd Entry

The snmpd entry specifies configuration parameters for the **snmpd** agent. See example 5 for a sample entry. An snmpd entry must be in the following format:

```
snmpd   Variable=Value
```

The = (equal sign) is absolutely required; there can be no spaces around it.

The following definitions apply to the `snmpd` entry:

Variable Specifies the specific configuration parameter. *Variable* can be one of the following values:

- **maxpacket**
- **querytimeout**.

Value Specifies the value of the specific variable.

The configurable variables and allowable values are:

maxpacket The maximum packet size, in bytes, that the **snmpd** agent will transmit. The minimum to which this variable can be set is 300 bytes. The maximum value to which this variable can be set is 56KB. If there is no snmpd entry for **maxpacket**, the system socket default levels will be used.

querytimeout The time interval in seconds at which the **snmpd** agent will query the interfaces to check for interface status changes. The minimum value to which **querytimeout** can be set is 30 seconds. If 0 is specified, **snmpd** will not query the interfaces for status changes. If there is no snmpd entry for **querytimeout**, the default value of 60 seconds is used.

The = (equal sign) is absolutely required; there can be no white space around it. There are no restrictions on the order in which the variables are entered in the snmpd entry. An snmpd entry can contain single or multiple variables.

The **snmpd** command sets the snmpd specific parameters immediately upon reading them. If the values are invalid, **snmpd** ignores them. If **snmpd** encounters an invalid field in the entry, processing is terminated for that entry and the **snmpd** command logs an EXCEPTIONS message if logging is enabled.

smux Entry

The smux entry specifies configuration information for SMUX associations between the **snmpd** agent and SMUX peer clients. See example 6 for a sample entry. A smux entry must be in the following format:

```
smux   ClientOID Password  IPAddress  NetMask
```

The following definitions apply to the `smux` entry:

ClientOID	Specifies the unique object identifier in dotted numeric notation of the SMUX peer client. The *ClientOID* must match the *ObjectID* specified in the **/etc/snmpd.peers** file.
Password	Specifies the password that the **snmpd** agent requires from the SMUX peer client to authenticate the SMUX association. The *Password* must match the *Password* in the **/etc/snmpd.peers** file.
IPAddress	The hostname or IP address in dotted notation of the host on which the SMUX peer client is executing.
NetMask	Specifies a network mask in dotted decimal notation for the specified hostname or IP address.

The minimum specification for the `smux` entry is:

```
smux   ClientOID   Password
```

The default values for this minimum smux entry are:

IPAddress	`127.0.0.1`
NetMask	`255.255.255.255`

Fields to the right of the minimum entry are optional, with the limitation that no fields to the left of a specified field are omitted. Any information to the right of *NetMask* is ignored. If no password is specified, there is no confirmation for the SMUX association. If neither the *IPAddress* nor *NetMask* are specified, the SMUX association is limited to the local host.

In the case of an invalid smux entry, the **snmpd** agent logs an EXCEPTIONS message if logging is enabled and the **snmpd** command ignores that smux entry.

sysLocation and sysContact Entry

The sysLocation and sysContact entries specify the values of the **sysLocation** and **sysContact** variables. The entry is specified in the following format:

```
sysLocation  "Austin, Texas, USA, XYZ, Bld 905, 5C-11"
sysContact   "Bill Roth, Amber Services, 1-512-849-3999"
```

The first part of the entry specifies the variable to be set, **sysLocation** or **sysContact**. The second part is a quoted character string representing the variable's value. The length of this string should not exceed 256 characters. If more than one entry is in the file, the last entry is used to define the variable. If there is not an entry for a particular variable, the value is defined to be the NULL string. If there is not a quoted string after the variable name, the first word on the line is used as the value. If there is nothing after the variable name, the NULL string is assumed.

The **snmpd** daemon uses the defined configuration file, whether it is the default file or specified from the command line, to save and read variables. The daemon does not need to be refreshed to get these new variables.

Note: Since these variables are settable, the **snmpd** daemon writes to the configuration file to update these variables on a set request. If you are editing the file and a set request changes the variables, the set request could be lost when the edited file is saved. This can be avoided by shutting down the daemon to change the configuration file, or by using the **snmpinfo** command to set the variable through normal methods.

Comments are specified by a # (pound sign) character and can be located anywhere in the **snmpd.conf** file. A comment begins at the # character and continues to the end of the line.

Note: It does not matter in which order the specific configuration entries for community, traps, views, logging, snmpd, and smux are placed in the **snmpd.conf** file. There is no order dependency for the various entries.

Examples

1. Example of community entries in the **snmpd.conf** file:

```
#  Community specifications
community public
community private  192.100.154.7  255.255.255.255    readWrite   1.17.2
community monitor  192.100.154.1  255.255.255.0      readWrite   1.17.2
community private  oilers
community simple   giants
community test     0.0.0.0        0.0.0.0            none
community nobody   0.0.0.0        255.255.255.255    readWrite   1.17.35
```

   The first entry exemplifies the minimum required specification for a community entry. The IP address defaults to 0.0.0.0. The network mask defaults to 0.0.0.0. The permissions default to readOnly. The view defaults to the entire MIB tree. This configuration enables the **snmpd** agent to accept all readOnly requests under the community name `public` regardless of the IP address. Write or set requests are rejected.

   The second entry limits the **snmpd** agent to accept `readWrite` requests under the community name `private` only from IP address `192.100.154.7` for MIB variables that are associated with the view name `1.17.2`.

   The third entry enables the **snmpd** agent to accept `readWrite` requests under the community name `monitor` from all IP addresses that start with `192.100.154`, as indicated by the network mask, for all MIB variables that are associated with the view name `1.17.2`.

The fourth entry sets the network mask to the default `255.255.255.255` and the permissions to the default, readOnly. This configuration enables the **snmpd** agent to accept readOnly requests under the community name `private` from the host named `oilers` for the entire MIB tree. The reuse of the community name `private` is independent of the usage in the second example entry.

The fifth entry sets the network mask to the default `255.255.255.255` and the default permissions to readOnly. This configuration enables the **snmpd** agent to accept readOnly requests for the entire MIB tree under the community name `simple` only from the host `giants`. Write or set requests are rejected.

The sixth entry causes the **snmpd** agent to reject all requests under the community name `test`, regardless of the IP address, because of the permission restriction of `none`.

The seventh entry causes the **snmpd** agent to reject all requests under the community name `nobody` because the network mask limits the IP address to entry `0.0.0.0`, which is reserved and not available for a host.

2. Example of trap entries in the **snmpd.conf** file:

```
#  Trap host notification specifications
trap traps   192.100.154.7
trap traps   129.35.39.233
trap events  giants
trap public  oilers  1.2.3       be
trap private 129.35.42.2101.2.4        7e
```

The first entry specifies that the **snmpd** agent is to notify the host with IP address `192.100.154.7` of all traps generated. The community name embedded in the trap packet will be `traps`.

The second entry specifies that the **snmpd** agent is to notify the host with IP address `129.35.39.233` of all traps generated. The community name embedded in the trap packet will be `traps`.

The third entry specifies that the **snmpd** agent is to notify the host `giants` of all traps generated. The community name embedded in the trap packet will be `events`.

The fourth entry specifies that the **snmpd** agent is to notify the host `oilers` of all traps generated except for the *warmStart* trap. The community name embedded in the trap packet will be `public`. The *ViewName*,`1.2.3`, is ignored.

The fifth entry specifies that the **snmpd** agent is to notify the host `129.35.42.210` of all traps generated except the *coldStart* trap. The community name embedded in the trap packet will be `private`. The *ViewName*, `1.2.4`, is ignored.

3. Examples of view entries in the **snmpd.conf** file:

```
#     View specifications
view  1.17.2 system enterprises view
view  1.17.35
view  2.10.1 iso.3
```

The first entry associates the view name `1.17.2` with the system, enterprises, and view MIB groups. A community name that is associated with view `1.17.2` will only be associated with the MIB variables in these three groups. Thus, a host that has read permissions with this community name association can only get values for MIB variables in these specified groups.

The second and third entries configure the **snmpd** agent to allow access to the entire MIB tree for hosts that have access privileges associated with these specified view names.

4. Examples of logging entries in the **snmpd.conf** file:

```
#   Logging specifications
logging    /tmp/snmpdlog enabled
logging    level=2   size=100000
```

These logging entries configure the **snmpd** agent to log messages at debug level `2` and below to the file named `/tmp/snmpdlog`. The size parameter limits the file size of the **/tmp/snmpd** log file to `100,000` bytes. When the log file reaches 100,000 bytes, the log file is rotated such that the full file is renamed to **/tmp/snmpdlog.0** and the new log file is named **/tmp/snmpdlog**.

5. Example of snmpd entries in the **snmpd.conf** file:

```
#       snmpd parameter specifications
snmpd   maxpacket=2048
snmpd   querytimeout=120
```

The first snmpd entry limits the size of packets transmitted by the **snmpd** agent to `2048` bytes.

The second entry sets the `querytimeout` parameter to `120` seconds. This configures the **snmpd** agent to query all the interfaces known to the TCP/IP kernel every two minutes for status changes.

6. Examples of smux entries in the **snmpd.conf** file:

```
#    smux configuration
smux 1.3.6.1.4.1.2.3.1.2.2       #gated
```

This smux entry configures the **snmpd** agent to allow the SMUX association only the **gated** SMUX peer client with no authentication. The SMUX peer must be running on the local host.

```
#    smux configuration
smux 1.3.6.1.4.1.2.3.1.2.2   private  #gated
```

This smux entry configures the **snmpd** agent to allow the SMUX association only the **gated** SMUX peer client having the password `private`. The SMUX peer must be running on the local host.

```
#    smux configuration
smux 1.3.6.1.4.1.2.3.1.2.2 private 0.0.0.0  0.0.0.0
```

This smux entry configures the **snmpd** agent to allow the SMUX association only the **gated** SMUX peer client having the password `private`. The SMUX peer can be running on any host.

```
#    smux configuration
smux 1.3.6.1.4.1.2.3.1.2.2 private 192.100.154.7 255.255.255.255
```

This smux entry configures the **snmpd** agent to allow the SMUX association only the **gated** SMUX peer client having the password`private`. The gated SMUX peer must be running on the host with IP address `192.100.154.7`

```
#    smux configuration
smux 1.3.6.1.4.1.2.3.1.2.2 private 192.100.154.1 255.255.255.0
```

This entry configures the **snmpd** agent to allow the SMUX association only the **gated** SMUX peer client having the password `private`. The gated SMUX peer can be running on any host in the network defined by `192.100.154.`

> **Note:** The SMUX peer client object identifier must be unique. Only *one* form of the preceding examples of smux entries for the gated SMUX peer client can be in the **snmpd.conf** file.

7. Example of sysLocation and sysContact entries in the **snmpd.conf** file:

```
# Definitions for sysLocation and sysContact
sysLocation   "Austin, Texas, USA, XYZ, Bld 905, 5C-11"
sysContact   "Bill Roth, Amber Services, 1-512-849-3999"
```

These entries set the value for the **sysLocation** and **sysContact** variables.

Implementation Specifics

This file is part of Simple Network Management Protocol Agent Applications in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

The **snmpd** command.

The **gated** daemon.

Problem Determination for the SNMP Daemon, Trap Processing, Understanding the SNMP Daemon Logging Facility in *AIX Version 4.3 System Management Guide: Communications and Networks*.

Understanding the SNMP

socks5c.conf File

Purpose

Contains mappings between network destinations and SOCKSv5 servers.

Description

The **/etc/socks5c.conf** file contains basic mappings (between network destinations, hosts or networks, and SOCKSv5 servers) to use when accessing network destinations. It is an ASCII file that contains records for server mappings. Text that follows a pound character (#) is ignored until the end of the line. Each record is on a single line in the following format:

<destination>[/*<prefixlength>*] *<server>*[:*<port>*]

You must separate the fields with whitespace. Records are separated by new line characters. The fields and modifiers in a record have the following values:

destination	Specifies a network destination. The *destination* variable may be either a name fragment or a numeric address (with optional *prefixlength*). If *destination* is an address, it may be either IPv4 or IPv6.
prefixlength	If specified, indicates the number of leftmost (network order) bits of an address to use when comparing to this record. It is valid only if *destination* is an address. If not specified, all bits are used in comparisons.
server	Specifies the SOCKSv5 server associated with *destination*. If *server* is **NONE** (must be all uppercase), this record indicates that target addresses matching *destination* should not use any SOCKSv5 server; instead, it should be contacted directly.
port	If specified, indicates the port to use when contacting *server*.

If a name fragment *destination* is present in **/etc/socks5c.conf**, all target addresses in SOCKSv5 operations will be converted into hostnames for name comparison (in addition to numeric comparisons with numeric records). The resulting hostname is considered to match if the last characters in the hostname match the specified name fragment.

When using this configuration information to determine the address of the appropriate SOCKSv5 server for a target destination, the *best* match is used. The *best* match is defined as follows:

If *destination* is numeric:	The most bits in the comparison (i.e., largest *prefixlength*)
If *destination* is a name fragment:	The most characters in the name fragment.

When both name fragment and numeric addresses are present, all name fragment entries are *better* than numeric address entries.

The following two implicit records are assumed as defaults for all destinations not specified in **/etc/socks5c.conf**.:

```
0.0.0.0/0 NONE #All IPv4 destinations; no associated server.
```

```
::/0      NONE #All IPv6 destinations; no associated server.
```

Security

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

Examples

```
#Sample socks5c.conf file
```

```
9.0.0.0/8    NONE    #Direct communication with all hosts in the 9 network.
```

```
129.35.0.0/16    sox1.austin.ibm.com
```

```
ibm.com    NONE    #Direct communication will all hosts matching "ibm.com" (e.g. "aguila.austin.ibm.com")
```

Implementation Specifics

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

Related Information

The **sock5tcp_connect** subroutine.

.srf File

Purpose

Contains all the text components with hypertext information embedded.

Description

The **.srf** file is one of several intermediate files produced for each document by InfoCrafter. The **.srf** file is a binary file that contains all the text components with hypertext link information embedded.

Implementation Specifics

This file is part of the InfoCrafter product.

Files

.srf	Contains text components with embedded linking information.

Related Information

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 (**auditcat**, **auditpr**, and, **auditselect**), 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 write (w) access to the root user only.

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:

   ```
   /usr/sbin/auditstream | /usr/sbin/auditselect -e \
    "result == FAIL" |/usr/sbin/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:

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

   This command allows timely auditing of user authentication events.

Implementation Specifics

This file is part of 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 audit events for audited objects (files).
/etc/security/audit/bincmds	Contains **auditbin** backend commands.

Related Information

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

Setting Up Auditing in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

Auditing Overview,

sysck.cfg File

Purpose

Contains file definitions for the trusted computing base.

Description

Note: The **sysck** command does not update this file. It is only updated by the **tcbck** command.

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.

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 in *AIX Version 4.3 System User's Guide: Operating System and Devices*

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 the *Flag*, *Flag* ..., *PBits* parameters. 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.

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

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

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

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`.

Since device configuration and the **sysck.cfg** database are independent and are not integrated, there is no automatic addition of **syck.cfg** entries when a device is added. Hence, given the automatic configuration of devices at boot time, it is the responsibility of the administrator to maintain **/etc/security/sysck.cfg**. This is also true in the case of mirrored rootvg, since **/dev/ipldevice** gets relinked dynamically to the other disk when the system is rebooted off the mirrored disk.

Security

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

2. A typical mode specification looks like the following example for a program that is part of the trusted computing base, that is a trusted process, and that has the **setuid** attribute enabled:

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

   OR

   ```
   mode = SUID,TP,TCB,750
   ```

Implementation Specifics

This file is part of 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, **tcbck** command, **usrck** command.

Access Control Lists in *AIX Version 4.3 System User's Guide: Operating System and Devices*

Security Administration

Temporary (TM.*) Files for BNU

Purpose

Store data files during transfers to remote systems.

Description

The Basic Networking Utilities (BNU) 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 **/var/spool/uucp/***SystemName*. The *SystemName* directory is named for the computer 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:

/var/spool/uucp/*SystemName***/TM.***xxPID***.000**

where the *SystemName* directory is named for the computer 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp/Systems file	Describes accessible remote systems.
/var/spool/uucp/*SystemName* directory	Contains BNU command, data, and execute files.
/var/spool/uucppublic/* directories	Contain files that BNU has transferred.
/var/spool/uucp/*SystemName***/D.*** files	Contain data to be transferred.

Related Information

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

The **uucico** daemon.

updaters File for NIS

Purpose

Updates NIS maps.

Description

The **/var/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. All items are followed by a new line except for actual bytes of key and actual bytes of data. The information passed is described below:

- 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 determines whether the user is allowed to make the change. If the user is not allowed, the **update** 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 `0`. If any errors exist that may prevent the **updaters** file from making the change, the command 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 (BOS) Runtime.

Related Information

The **publickey** file.

The **update** command.

The **ypupdated** daemon.

Checklist for Administering Secure NFS, Network File System (NFS) Overview for System Management,

user File

Purpose

Contains extended user attributes.

Description

The **/etc/security/user** file contains extended user attributes. This is an ASCII file that contains attribute stanzas for users. The **mkuser** command creates a stanza in this file for each new user and initializes its attributes with the default attributes defined in the **/usr/lib/security/mkuser.default** file.

Each stanza in the **/etc/security/user** file 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. You can have multiple default stanzas in the **/etc/security/group** file. A default stanza applies to all of the stanzas that follow, but does not apply to the stanzas preceding it.

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.

Attributes

If you have the proper authority, you can set the following user attributes:

account_locked
Indicates if the user account is locked. Possible values include:

- **true** The user’s account is locked. The values **yes**, **true**, and **always** are equivalent. The user is denied access to the system.
- **false** The user’s account is not locked. The values **no**, **false**, and **never** are equivalent. The user is allowed access to the system.

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

- **true** The user is an administrator. Only the root user can change the attributes of users defined as administrators.
- **false** The user is not an administrator. This is the default value.

admgroups
Lists the groups the user administrates. The *Value* parameter is a comma-separated list of group names.

auditclasses
Lists the user’s audit classes. The Value parameter is a list of comma-separated classes, or a value of **ALL** to indicate all audit classes.

auth1
Lists the primary methods for authenticating the user. The *Value* parameter is a comma-separated list of *Method***;***Name* pairs. The *Method* parameter is the name of the authentication method. The *Name* parameter is the user to authenticate. If you do not specify a *Name* parameter, the name of the invoking login program is used.

Valid authentication methods are defined in the **/etc/security/login.cfg** file. The SYSTEM method is always used in addition to the methods listed here on **auth1**, even if SYSTEM is not specified. The SYSTEM method is defined by the SYSTEM user attribute. If you do not want the user to authenticate using the SYSTEM method, specify NONE for the SYSTEM user attribute.

auth2
Lists the secondary methods used to authenticate the user. The *Value* parameter is a comma-separated list of *Method***;***Name* pairs. The *Method* parameter is the name of the authentication method. The *Name* parameter is the name of the user to be authenticated.

If this attribute is not specified, the default is NONE, indicating that no secondary authentication check is made. Valid authentication methods are defined in the **/etc/security/login.cfg** file. If you do not specify a *Name* parameter, the name of the invoking login program is used.

daemon
Indicates whether the user specified by the Name parameter can execute programs using the **src** (system resource controller) daemon. Possible values are:

true The user can initiate **src** sessions. This is the default.

false The user cannot initiate **src** sessions.

dce_export
Allows the DCE registry to overwrite the local user information with the DCE user information during a DCE export operation. Possible values are:

true Local user information will be overwritten.

false Local user information will not be overwritten.

dictionlist
Defines the password dictionaries used by the composition restrictions when checking new passwords.

The password dictionaries are a list of comma-separated absolute path names, evaluated from left to right. All dictionary files and directories must be write-protected from all users except root. The dictionary files are formatted one word per line. The word starts in the first column and terminates with a new-line character. Only 7-bit ASCII words are supported for passwords. If you install text processing on your system, the recommended dictionary file is the **/usr/share/dict/words** file.

expires
Identifies the expiration date of the account. The Value parameter is a 10-character string in the *MMDDhhmmyy* form, where *MM* = month, *DD* = day, *hh* = hour, *mm* = minute, and *yy* = last 2 digits of the years 1939 through 2038. All characters are numeric. If the *Value* parameter is 0, the account does not expire. The default is 0. See the **date** command for more information.

histexpire
Defines the period of time (in weeks) that a user cannot reuse a password. The value is a decimal integer string. The default is 0, indicating that no time limit is set.

histsize
Defines the number of previous passwords a user cannot reuse. The value is a decimal integer string. The default is 0.

login
Indicates whether the user can log in to the system with the **login** command. Possible values are:

true The user can log in to the system. This is the default.

false The user cannot log in to the system.

logintimes
Specifies the times, days, or both the user is allowed to access the system. The value is a comma-separated list of entries of the following form:

```
[!]:time-time
        -or-
[!]day[-day][:time-time]
        -or-
[!]date[-date][:time-time]
```

The *day* variable must be one digit between 0 and 6 that represents one of the days of the week. A 0 (zero) indicates Sunday and a 6 indicates Saturday.

The *time* variable is 24-hour military time (1700 is 5:00 p.m.). Leading zeroes are required. For example, you must enter `0800,` not `800`. The *time* variable must be four characters in length, and there must be a leading colon (:). An entry consisting of only a time specification applies to every day. The start hour of a time value must be less than the end hour.

The *date* variable is a four digit string in the form *mmdd*. *mm* represents the calendar month and *dd* represents the day number. For example `0001` represents January 1. *dd* may be `00` to indicate the entire month, if the entry is not a range, or indicating the first or last day of the month depending on whether it appears as part of the start or end of a range. For example, `0000` indicates the entire month of January. `0600` indicates the entire month of June. `0311-0500` indicates April 11 through the last day of June.

Entries in this list specify times that a user is allowed or denied access to the system. Entries not preceded by an exclamation point (`!`) allow access and are called ALLOW entries. Entries prefixed with an exclamation point (`!`) deny access to the system and are called DENY entries. The `!` operator applies to only one entry, not the whole restriction list. It must appear at the beginning of an entry.

loginretries Defines the number of unsuccessful login attempts allowed after the last successful login before the system locks the account. The value is a decimal integer string. A zero or negative value indicates that no limit exists. Once the user's account is locked, the user will not be able to log in until the system administrator resets the user's `unsuccessful_login_count` attribute in the **/etc/security/lastlog** file to be less than the value of **loginretries**. To do this, enter the following:

```
chsec -f /etc/security/lastlog -s username -a \
unsuccessful_login_count=0
```

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.

maxexpired Defines the maximum time (in weeks) beyond the **maxage** value that a user can change an expired password. After this defined time, only an administrative user can change the password. The value is a decimal integer string. The default is -1, indicating no restriction is set. If the **maxexpired** attribute is 0, the password expires when the **maxage** value is met. If the **maxage** attribute is 0, the **maxexpired** attribute is ignored.

maxrepeats Defines the maximum number of times a character can be repeated in a new password. Since a value of 0 is meaningless, the default value of 8 indicates that there is no maximum number. 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 were not in the old password. The value is a decimal integer string. The default is a value of 0, indicating no minimum number.

minlen Defines the minimum length of a password. The value is a decimal integer string. The default is a value of 0, indicating no minimum length. The maximum value allowed is 8. This attribute is determined by the **minalpha** attribute added to the **minother** attribute. If the result of this addition is greater than the **minlen** attribute, the minimum length is set to the result.

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.

pwdchecks Defines the password restriction methods enforced on new passwords. The value is a list of comma-separated method names and is evaluated from left to right. A method name is either an absolute path name or a path name relative to **/usr/lib** of an executable load module.

pwdwarntime Defines the number of days before the system issues a warning that a password change is required. The value is a decimal integer string. A zero or negative value indicates that no message is issued. The value must be less than the difference of the **maxage** and **minage** attributes. Values greater than this difference are ignored, and a message is issued when the **minage** value is reached.

registry Defines the authentication registry where the user is administered. It is used to resolve a remotely administered user to the local administered domain. This situation may occur when network services unexpectedly fail or network databases are replicated locally. Example values are **files** or **NIS** or **DCE**.

rlogin Permits access to the account from a remote location with the **telnet** or **rlogin** commands. Possible values are:

- **true** The user account can be accessed remotely. This is the default **rlogin** value.
- **false** The user cannot be accessed remotely.

su Indicates whether another user can switch to the specified user account with the **su** command. Possible values are:

- **true** Another user can switch to the specified account. This is the default.
- **false** Another user cannot switch to the specified account.

sugroups Lists the groups that can use the **su** command to switch to the specified user account. The *Value* parameter is a comma-separated list of group names, or a value of **ALL** to indicate all groups. An ! (exclamation point) in front of a group name excludes that group. If this attribute is not specified, all groups can switch to this user account with the **su** command.

SYSTEM Defines the system authentication method for the user. The SYSTEM method is always used to authenticate the user, no matter what other methods are specified on the **auth1** and **auth2** attributes. If you do not want the user to authenticate using the SYSTEM method, specify NONE. The AIX TCP socket daemons (that is, ftpd, rexecd, rshd) do not use these authentication methods. Instead, they access the **passwd** file directly. So, if SYSTEM is set to NONE, authentication is turned off for all of these commands. Specify the value for SYSTEM using the following grammar:

```
"SYSTEM"        ::= EXPRESSION
EXPRESSION      ::= PRIMITIVE  |
                     "("EXPRESSION")"  |
                      EXPRESSION OPERATOR EXPRESSION
PRIMITIVE       ::= METHOD  |
                     METHOD "["RESULT"]"
RESULT          ::= "SUCCESS" | "FAILURE" | "NOTFOUND" |
                    "UNAVAIL"  | "*"
OPERATOR        ::= "AND" | "OR"
METHOD          ::= "compat" | "files" | "NONE" |
                    [a-z,A-Z,0-9]*
```

An example of the syntax is:

```
SYSTEM = "DCE OR DCE[UNAVAIL] AND
compat"
```

tpath Indicates the user's trusted path status. The possible values are:

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

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

nosak The secure attention key (SAK) is disabled for all processes run by the user. Use this value if the user transfers binary data that may contain the SAK sequence. This is the default value.

on The user has normal trusted path characteristics and can invoke a trusted path (enter a trusted shell) with the secure attention key (SAK).

ttys Lists the terminals that can access the account specified by the *Name* parameter. The *Value* parameter is a comma-separated list of full path names, or a value of **ALL** to indicate all terminals. The values of **RSH** and **REXEC** also can be used as terminal names. An ! (exclamation point) in front of a terminal name excludes that terminal. If this attribute is not specified, all terminals can access the user account. If the *Value* parameter is not **ALL**, then `/dev/pts` must be specified for network logins to work.

umask	Determines file permissions. This value, along with the permissions of the creating process, determines a file's permissions when the file is created. The default is 022.

Changing the user File

You should access this file through the commands and subroutines defined for this purpose. You can use the following commands to change the **user** file:

- **chuser**
- **lsuser**
- **mkuser**
- **rmuser**

The **mkuser** command creates an entry for each new user in the **/etc/security/user** file and initializes its attributes with the attributes defined in the **/usr/lib/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 a user, 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	file name

Examples

1. 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
```

2. To allow all ttys except `/dev/tty0` to access the user account, change the ttys entry so that it reads as follows:

```
ttys = !/dev/tty0,ALL
```

Implementation Specifics

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

Files

/etc/group	Contains the basic group attributes.
/etc/passwd	Contains the basic user attributes.
/etc/security/audit/config	Contains audit system configuration information.
/etc/security/environ	Contains the environment attributes of users.
/etc/security/group	Contains the extended attributes of groups.
/etc/security/limits	Contains the process resource limits of users.
/etc/security/login.cfg	Contains configuration information for user log in and authentication.
/etc/security/passwd	Contains password information.
/usr/lib/security/mkuser.default	Contains default user configurations.
/etc/security/user	Contains extended user attributes.
/etc/security/lastlog	Contains last login information.

Related Information

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

The **enduserdb** subroutine, **getuserattr** 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 Administration and Managing

user.roles File

Purpose

Contains the list of roles for each user. This system file only applies to AIX Version 4.2.1 and later.

Description

The **/etc/security/user.roles** file contains the list of roles for each user. This is an ASCII file that contains a stanza for system users. Each stanza is identified by a user name followed by a : (colon) and contains attributes in the form *Attribute=Value*. Each attribute pair ends with a newline character as does each stanza.

This file supports a default stanza. If an attribute is not defined, either the default stanza or the default value for the attribute is used.

A stanza contains the following attribute:

roles	Contains the list of roles for each user.

For a typical stanza, see the "Examples" section.

Typically, the **/etc/security/user.roles** stanza contains an entry for every user and a list of data associated with that user. The roles database does not require an entry per user. The size of each entry is one line.

The **user.roles** file is kept separately from the **/etc/security/user** file for performance reasons. Several commands scan this database, so system performance increases with smaller files to scan (especially on systems with large numbers of users).

Changing the user.roles File

You should access this file through the commands and subroutines defined for this purpose. You can use the following commands to change the **user.roles** file:

- **chuser**
- **lsuser**
- **mkuser**

The **mkuser** command creates an entry in the **/etc/security/user.roles** file for each new user when the **roles** attribute is used. To change the attribute values, use the **chuser** command with the **roles** attribute. To display the attributes and their values, use the **lsuser** command with the **roles** attribute.

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

Security

Access Control: This file grants read and write access to the root user, and read access to members of the security group.

Examples

A typical stanza looks like the following example for the `username` role:

```
username:
        roles = role1,role2
```

Implementation Specifics

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

Files

/etc/security/roles	Contains the list of valid roles.
/etc/security/user.roles	Contains the list of roles for each user.
/etc/security/smitacl.group	Contains the group ACL definitions.
/etc/security/smitacl.user	Contains the user ACL definitions.

Related Information

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

vfs File

Purpose

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

Description

The **/etc/vfs** file describes the virtual file systems 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 are examples of the three types of lines in the **vfs** file:

- Comments

  ```
  # This is a comment.
  # Comments begin with a # (pound sign).
  # Blank lines are ignored.
  # The following example 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               /sbin/helpers/v3fshelper
  nfs      2     /etc/nfsmnthelp    none
  cdrfs    5     none               none
  ```

The comments are in text for explanatory purposes. The general control lines, which are designated by a % (percent) character, configure the actions of the following commands:

- **mount**
- **umount**
- **mkfs**
- **fsck**
- **fsdb**

- **df**
- **ff**

For example, a line like `%defaultvfs` indicates the default local virtual file system is used if no VFS is specified by the **mount** command or in the **/etc/filesystems** file. The entry is the name of the VFS as indicated in the file. If a second entry is listed on the same line, it is taken to be the default remote VFS. The `%defaultvfs` control line may leave off the remote VFS specification.

The VFS 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 VFS.
`mnt_helper`	Path name of the mount helper program of this VFS. 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 **/sbin/helpers** directory.
`fs_helper`	Path name of the file system helper program of this VFS. 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 **/sbin/helpers** directory.

Implementation Specifics

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

Files

/etc/filesystems	Lists the known file systems and defines their characteristics.

Related Information

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

The File Systems Overview for System Management in *AIX Version 4.3 System Management Concepts: Operating System and Devices*

xferstats File for BNU

Purpose

Contains information about the status of file transfer requests.

Description

The **/var/spool/uucp/.Admin/xferstats** file contains information about the status of each Basic Networking Utilities (BNU) file transfer request. The **xferstats** file contains:

- 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

Examples

Following is a typical entry in the **xferstats** file:

```
zeus!jim M (10/11-16:10:33) (C,9234,1) [-] -> 1167 / 0.100secs
```

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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/var/spool/uucp/.Admin directory	Contains the **xferstats** file and other BNU administrative files.

Related Information

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

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

xtab File for NFS

Purpose

Contains entries for currently exported NFS directories.

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** flag of the **exportfs** command.

Implementation Specifics

This file is part of NFS in Network Support Facilities in Base Operating System (BOS) 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.

Chapter 2. File Formats

Certain files in the 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.

.3270keys File Format for TCP/IP

Purpose

Defines keyboard mapping and colors for the **tn** and **telnet** command.

Description

The **$HOME/.3270keys** file specifies for a user a **tn** or **telnet** command key mapping that differs from the default mapping found in the **/etc/3270.keys** file. You can use it, for example, to make the Action key act as the Enter key.

If you are using a color display, you can also customize the colors for various 3270 display attributes by setting attributes in the **.3270keys** file. The default mapping in the **/etc/3270.keys** file is generic. The user can also load the user-defined files for specific terminal types by using the **.3270keys** file. The **.3270keys** file is specified in the user’s home directory. The default background color is black. You cannot configure the background color.

The **tn** or **telnet** command first checks the **$HOME** directory for the **.3270keys** file and loads it. If the file doesn’t exist, the **/etc/3270.keys** file is loaded. Both files, by default, end with an `if` statement and a list of terminal types. If the **TERM** environment variable matches one of the listed terminals, a second file is loaded. If the **TERM** variable does not match, the **tn** or **telnet** command uses the generic key bindings specified before the `if` statement and prints the message `NOBINDINGS`.

> **Note:** When remapping keys to customize your **$HOME/.3270keys** file, remember that you cannot map a 3270 function to the Esc key alone. You can specify the Esc key only in combination with another key. Also, when mapping keys, do not duplicate key sequences. For example, if you have mapped the backtab function to the ^A (the Ctrl-A key sequence), then mapping the PF1 function key to ^Aep (the Ctrl-Aep key sequence) is going to conflict with the backtab mapping.

The $HOME/.3270keys.hft File

You can also use the **/usr/lpp/tcpip/samples/3270keys.hft** sample file to create a **$HOME/.3270keys.hft** file by copying the sample file to your home directory and modifying it as necessary.

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; for example, ^M.

~ Set high-order bit for next character.

The following are valid colors for 3270 display attributes:

- black
- blue
- red
- green
- white
- magenta
- cyan

For more information about changing the assignment of a key set, see "Changing the Assignment of a Key Set" in *AIX Version 4.3 System User's Guide: Communications and Networks.*

> **Note:** The **3270keys.hft** file supports the Attention key, which sends an IAC BREAK TELNET protocol sequence to the TELNET server on a VM or MVS system. The TELNET server is responsible for implementing the Attention key. Example 2 shows the format for binding the Attention key to the Ctrl-F12 key sequence.

Examples

1. The following example binds the Backspace key and the Tab keys:

   ```
         3270 Function  Sequence  Key
   bind   backspace     "\b"      #backspace key
   bind   tab           "\t"      #tab key
   ```

   The # (pound sign) is used to indicate comments.
2. The following example binds the Attention key to the Ctrl-F12 key sequence:

   ```
         3270 Function  Sequence  Key
   bind   attention     "\e[036q" #attention key
   ```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/etc/3270.keys	Contains the default keyboard mapping for non-HFT keyboards.
/etc/3270keys.hft	Contains the default keyboard mapping for HFT keyboards.
/usr/lpp/tcpip/samples/3270keys.hft	Contains a sample HFT keyboard mapping.

Related Information

The **telnet**, **tn**, or **tn3270** command.

The **map3270** file format.

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 **/usr/include/sys/acct.h** file defines the records in these files, which are written when a process exits.

The acct structure

The **acct** structure in the **acct.h** header file contains the following fields:

`ac_flag` Specifies one of the following accounting flags 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` Specifies the exit status. A flag that indicates how the process terminated.

`ac_uid` Specifies the user ID of the process for which the accounting record is written.

`ac_gid` Specifies the group ID of the process for which the accounting record is written.

`ac_tty` Specifies the terminal from which the process was started.

`ac_btime` Specifies the beginning time. The time at which the process started.

`ac_utime` Specifies the amount of user time, in seconds, used by the process.

`ac_stime` Specifies the amount of system time, in seconds, used by the process.

`ac_etime` Specifies the amount of time, in seconds, elapsed since the command ran.

`ac_mem` Specifies the average amount of memory used by the process. Every clock interrupt (or clock tick,100 times per second), the **sys_timer** routine is called to update the user data for the current process. If the process is in user mode, both its **u_utime** value and memory usage values are incremented; otherwise, only its **u_stime** value is incremented. The **sys_timer** routine calls the **vms_rusage** routine to obtain the kilobytes of real memory being used by TEXTSEG (#1), the PRIVSEG (#2), and the big-data segments (#3-11), if used. These values are added to the total memory usage value at each clock tick during which the process is not in kernel mode. When the process ends, the **acctexit** routine computes how many clock ticks occurred while the process executed (in both user and kernel modes) and divides the total memory usage value by this number to give an average memory usage for the process. This value is recorded as a two-byte unsigned short integer.

`ac_io` Specifies the number of characters transferred by the process.

`ac_rw` Specifies the number of blocks read or written by the process.

`ac_comm` Specifies 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.

The 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 */
};
```

Implementation Specifics

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

Related Information

The **acctcms** command, **acctcom** command, **acctcon1** or **acctcon2** command, **acctdisk** command, **acctmerg** command, **acctprc1**, **acctprc2**, or **accton** command, **runacct** command.

The **acct** subroutine, **fork** subroutine, **exec** subroutine.

The Header Files Overview in *AIX Version 4.3 Files Reference*.

ar File Format (Big)

Purpose

This document describes the default **ar** library archive format for AIX 4.3.

Description

The **ar** (archive) command combines several files into one. The **ar** command creates an archive file. The **ld** (link editor) command searches archive files to resolve program linkage. The **/usr/include/ar.h** file describes the archive file format.This file format accommodates both 32-bit and 64-bit object files within the same archive.

This is the default file format used by the **ar** command. To use a format portable to versions prior to AIX 4.3.0, refer to **ar File Format (Small)**.

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 __AR_BIG__
#define AIAMAGBIG "<bigaf>\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[20];      /*Offset to member table */
char  fl_gstoff[20];      /*Offset to global symbol table */
char  fl_gst64off[20];    /*Offset global symbol table for 64-bit objects */
char  fl_fstmoff[20];     /*Offset to first archive member */
char  fl_lstmoff[20];     /*Offset to last archive member */
char  fl_freeoff[20];     /*Offset to first mem on free list */

} ;
```

The 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 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, the 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, free list space 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[20];     /* File member size - decimal */
           char ar_nxtmem[20];   /* Next member offset-decimal */
           char ar_prvmem[20];   /* Previous member offset-dec */
           char ar_date[12];     /* File member date-decimal */
           char ar_uid[12];      /* File member userid-decimal */
           char ar_gid[12];      /* File member group id-decimal */
           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) + 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 format. Numeric information, with the exception of the `ar_mode` field, is stored as decimal numbers; the `ar_mode` field is stored in octal format. 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 quickly locates 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 member is 20 bytes long and stored in ASCII format as a decimal number.
- The array of offsets into the archive file. The length is 20 times the number of members. Each offset is 20 bytes long and stored in ASCII format as a decimal number.

- The name string table. The size is:

  ```
  ar_size - (20 * (the number of members +1));
  ```

  that is, the size equals the total length of all members minus the length of the offsets, minus the length of the number of members.

  The string table contains the same number of strings as offsets. All strings are null-terminated. Each offset from the array corresponds sequentially to a name in the string table.

Global Symbol Tables

Immediately following the member table, the archive file contains two global symbol tables. The first global symbol table locates 32-bit file members that define global symbols; the second global symbol table does the same for 64-bit file members. If the archive has no 32-bit or 64-bit file members, the respective global symbol table is omitted. The **strip** command can be used to delete one or both global symbol tables from the archive. The `fl_gstoff` field in the fixed-length header contains the offset to the 32-bit global symbol table, and the `fl_gst64off` contains the offset to the 64-bit global symbol table. The global symbol table members have zero-length names. The **ar** command automatically creates and updates, but does not list the global symbol tables. A global symbol table contains the following information:

- The number of symbols. This is 8 bytes long and can be accessed with the **sgetl** and **sputl** commands.
- The array of offsets into the archive file. The length is eight times the number of symbols. Each offset is 8 bytes long and can be accessed with the **sgetl** and **sputl** commands.
- The name-string table. The size is:

  ```
  ar_size - (8  *  (the number of symbols + 1));
  ```

  That is, the size equals the total length of the members, minus the length of the offsets, minus the length of the number of symbols.

  The string table contains the same number of strings as offsets. All strings are null-terminated. Each offset from the array corresponds sequentially to a name in the string table.

Related Information

The **a.out** file format.

The **ar** command, **ld** command, **strip** command.

The **sgetl** or **sputl** subroutine.

The Header Files Overview in *AIX Version 4.3 Files Reference*.

ar File Format (Small)

Purpose

Describes the small indexed archive file format, in use prior to AIX 4.3. This format is recognized by AIX commands for backward compatability purposes only. See **ar File Format (Big)** for the current archive file format.

Description

The **ar** (archive) command combines several files into one. The **ar** command creates an archive file. The **ld** (link editor) command searches archive files to resolve program linkage. The **/usr/include/ar.h** file describes the archive file format. This archive format only handles 32-bit XCOFF members. The **ar File Format (Big)** handles both 32-bit and 64-bit XCOFF members

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 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 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, the 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, free list space 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 - decimal */
        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) + 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 format. Numeric information, with the exception of the `ar_mode` field, is stored as decimal numbers; the `ar_mode` field is stored in octal format. 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 quickly locates 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 member is 12 bytes long and stored in ASCII format 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 format as a decimal number.
- The name string table. The size is:

```
ar_size - (12 * (the number of members +1));
```

that is, the size equals the total length of all members minus the length of the offsets, minus the length of the number of members.

The string table contains the same number of strings as 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 locates file members that define global symbols. The **strip** command deletes 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** and **sputl** commands.
- The array of offsets into the archive file. The length is four times the number of symbols. Each offset is 4 bytes long and can be accessed with the **sgetl** and **sputl** commands.
- The name-string table. The size is:

  ```
  ar_size - (4 * (the number of symbols + 1));
  ```

  That is, the size equals the total length of the members, minus the length of the offsets, minus the length of the number of symbols.

  The string table contains the same number of strings as offsets. All strings are null-terminated. Each offset from the array corresponds sequentially to a name in the string table.

Related Information

The **a.out** file format.

The **ar** command, **ld** command, **strip** command.

The **sgetl** or **sputl** subroutine.

The Header Files Overview in *AIX Version 4.3 Files Reference*.

ate.def File Format

Purpose

Determines default settings for the Asynchronous Terminal Emulation (ATE) program.

Description

The **ate.def** file sets the defaults for use in asynchronous connections and file transfers. This file is created in the current directory during the first run of ATE. The **ate.def** file contains the default values in 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

Edit the **ate.def** file with any ASCII text editor to permanently change the values of these characteristics. Temporarily change the values of these characteristics with the ATE **alter** and **modify** subcommands, accessible from either ATE Main Menu.

Parameters in the ate.def File

Type parameter names in uppercase letters in the **ate.def** file. Spell the parameters exactly as they appear in the original default file. Define only one parameter per line. An incorrectly defined value for a parameter causes ATE to return a system message. However, the program continues to run using the default value.

These are the **ate.def** file parameters:

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 the user selects 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 baud rate, or the number of bits transmitted per second (bps). The speed must match the speed of the modem and that of the remote system.

Options: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19,200.

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 the user autodials with a modem. For the proper dial commands, consult the modem documentation.

Options: ATDT, ATDP, or other values, depending on the type of modem.

Default: ATDT.

FINAL — Defines the dial suffix, a string that must follow the telephone number when the user autodials with a modem. For the proper dial commands, consult the modem documentation.

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 it is interrupted. If the `ATTEMPTS` parameter is set to 0, 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, 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**:

File transfer protocol 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 to detect data transmission errors and retransmit the data.

Options: **p** (**pacing**), **x** (**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 line-feed 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 file name less than 40 characters long.

Default: The **kapture** file.

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 sent returns and displays on the screen. When the `ECHO` parameter is on, each character is displayed twice: first when it is entered, and again when it returns over a connection. When the `ECHO` parameter is off, each character displays only 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 (usually the Ctrl-B 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 (usually the Ctrl-B 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).

MAINMENU_KEY — Defines the control key sequence that returns the Connected Main Menu so the user can issue a command during an active connection. The MAINMENU_KEY (usually the Ctrl-V key sequence) functions only from the connected state.

Options: Any ASCII control character.

Default: ASCII octal 026 (SYN).

PREVIOUS_KEY	Defines the control key sequence that displays the previous screen anytime during the program. The screen displayed varies, depending on the screen in use when the user presses PREVIOUS_KEY (usually the Ctrl-R key sequence). Options: Any ASCII control character. Default: ASCII octal 022 (DC2). The ASCII control character is mapped to the interrupt signal.

Notes:

1. Changing or remapping may be necessary if control keys conflict across applications. For example, if the control keys mapped for the ATE program conflict with those in a text editor, remap the ATE control keys.

2. The ASCII control character selected 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.

Examples

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 running the ATE program:

```
RATE         300
DEVICE       tty3
TRANSFER     x
DIRECTORY    my.dir
```

The time the ATE program starts from that directory, the program uses the defined values.

Implementation Specifics

This file is part of Asynchronous Terminal Emulation in BOS Extensions 2.

Files

/usr/lib/dir	Contains the default dialing directory file.

Related Information

The **ate** command.

The **alter** subcommand, **connect** subcommand, **directory** subcommand, **modify** subcommand, **receive** subcommand, **send** subcommand.

audit File Format

Purpose

Describes the auditing data structures.

Description

The **/usr/include/sys/audit.h** file contains structure and constant definitions for the auditing system commands, subroutines, and daemons:

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 nonzero 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 Class Format

The format of the audit class is described by the **audit_class** structure, which contains the following fields:

`ae_name` A pointer to the name of the audit class.

`ae_list` A pointer to a list of null-terminated audit event names for this audit class. The list is ended by a null name (a leading null byte or two consecutive null bytes).

Note: Event and class names are limited to 15 significant characters.

`ae_len` The length of the event list in the `ae_list` member. This length includes the terminating null bytes. On an **AUDIT_SET** operation, the caller must set this member to indicate the actual length of the list (in bytes) pointed to by `ae_list`. On an **AUDIT_GET** or **AUDIT_LOCK** operation, the **auditevents** subroutine sets this member to indicate the actual size of the list.

Audit Object Format

The format of the audit object is described by the **o_event** structure, which contains the following fields:

`o_type` Specifies the type of the object, in terms of naming space. Currently, only one object-naming space is supported:

AUDIT_FILE Denotes the file system naming space.

`o_name` Specifies the name of the object.

`o_event` Specifies any array of event names to be generated when the object is accessed. Note that event names in AIX are currently limited to 16 bytes, including the trailing null. The index of an event name in this array corresponds to an access mode. Valid indexes are defined in the **audit.h** file and include the following:

- **AUDIT_READ**
- **AUDIT_WRITE**
- **AUDIT_EXEC**

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_magic`	Magic number for audit record.
`ah_length`	The length of the tail portion of the audit record.
`ah_event[16]`	The name of the event and a null terminator.
`ah_result`	An indication of whether the event describes a successful operation. The values for this field are:

0	Indicates successful completion.
1	Indicates a failure.
>**1**	An **errno** value describing the failure.

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 in seconds at which this audit record was written.
`ah_ntime`	The nanoseconds offset from `ah_time`.

The record tail follows this header information.

Related Information

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

The **auditbin** daemon.

The **audit** subroutine, **auditbin** subroutine, **auditevents** subroutine, **auditlog** subroutine, **auditobj** subroutine, **auditproc** subroutine, **auditwrite** subroutine.

Header Files Overview in *AIX Version 4.3 Files Reference*.

bootptab File Format

Purpose

Default configuration database for the Internet Boot Protocol server (**bootpd**).

Description

The **bootpd** configuration file contains entries for clients that use the **bootpd** daemon to get boot information. This file may be modified using the System Management Interface Tool (SMIT) to configure an Xstation or a Diskless client or the file may be modified manually.

The client host information consists of case-sensitive tag symbols used to represent host parameters. These host parameter declarations are separated by : (colon). For example:

`HostName:`*`Tg=Value:Tg=Value:Tg=Value`*

where:

HostName	Specifies the name of a BOOTP client. This must always be the first field in the entry. The **bootpd** daemon attempts to send the entire host name as it is specified in this field. However, if the host name does not fit into the reply packet, the name is shortened to the host field (up to the first period, if present) and tried again. An arbitrarily truncated host name is never sent. If nothing reasonable fits, nothing is sent.

Tg Specifies a character tag. Some tags must be followed by an = (equal sign) and a *Value*. Other tags appear in a Boolean form with no value. The recognized tags are:

bf Specifies the boot file. The *Value* is an ASCII string that can be optionally surrounded by double quotes.

The client's request and the **hd** and **bf** tags determine how the server fills in the boot file field of the bootp reply packet.

- If the client's bootp request specifies an absolute path name and that file exists on the server machine, that path name is returned in the reply packet. If the file cannot be found, the request is discarded; no reply is sent.
- If the client's bootp request specifies a relative path name, a full path name is formed by adding the value of the **hd** tag and testing for existence of the file. If the **hd** tag is not supplied in the configuration file, or if the resulting boot file cannot be found, the request is discarded.
- If a client's bootp request specifies a null boot file, the server always replies with the boot file field filled according to the following guidelines:
  - If the **bf** tag gives an absolute path name and the file exists, that path name is returned in the reply packet.
  - If the **hd** and **bf** tags combined specify an accessible file, that file name is returned in the reply.
  - If a complete file name cannot be determined or the file does not exist, the reply contains a zeroed-out boot file field.

In all cases, the existence of a file means that, in addition to actually being present, the file must have its Public Read Access bit set for the **tftp** program to permit the file transfer. Also, all file names are tried as *FileName.HostName* first. Then, the file names are tried simply as *FileName*, thus providing for individual per-host boot files.

bs Specifies the boot file size. The *Value* is in 512-octet blocks, which can be decimal, octal, or hexadecimal integers, or the keyword `auto`, which causes the server to automatically calculate the boot file size at each request. Specifying the **bs** tag as a Boolean value is the same as specifying the keyword `auto` for *Value*.

bt This tag is obolete and is ignored.

cs Cookie server address list.

ds Domain name server address list.

dt Specifies that an old style boot be performed to this client. This tag is needed to do a broadcast boot on some Xstations.

gw Specifies the gateway address list. If this tag is defined, the **sm** (subnet mask) tag must also be defined.

ha Specifies the host hardware address. The *Value* is in hexadecimal with optional periods and/or a leading 0x. The **ha** tag must be preceded, explicitly or implicitly, by the **ht** tag.

hd Specifies the boot file home directory. The *Value* is an ASCII string that can be optionally surrounded by double quotes.

hn Specifies that the host name should be sent to clients. The **hn** tag is a Boolean tag and does not take the = (equals sign) and a *Value*.

ht Specifies the host network type. The *Value* is an unsigned decimal, octal, or hexadecimal integer, or a symbolic name. The **ht** tag must precede the **ha** tag, explicitly or implicitly.

The **bootpd** daemon supports the following network hardware types:

Network Type	Symbol(s)
10Mb Ethernet	ether or ethernet
IEEE 802.3 Ethernet	ieee802
IEEE 802.5 Token Ring	tr or tokenring
FDDI	fddi

Since the 10Mb Ethernet and the IEEE 802.3 Ethernet use the same network adapter, the hardware address for both networks is the same. In cases like this, the client can have more than one entry in the **/etc/bootptab** file on a designated boot host. However, each entry must have a different hostname for the client and a different host network type. In this case, one entry would have a host network type of ether and the other entry would have a host network type of ieee802. This allows the client to boot over either the 10Mb Ethernet or the IEEE 802.3 Ethernet.

ip Specifies the client's IP address. The *Value* is a single IP address in standard Internet dot notation. The IP address can use decimal, octal, or hexadecimal numbers (octal numbers begin with 0, hexadecimal numbers begin with 0x or 0X). The **ip** tag, used as a Boolean value, looks up the IP address using the host name field.

lg MIT-LCS UDP log server address list.

lp Line Printer Server (LPR) server address list.

ns IEN-116 name server address list.

rl Resource location protocol server address list.

sa Specifies the IP address of the TFTP server where the client's boot file resides. The *Value* is a single IP address in standard Internet dot notation. The IP address can use decimal, octal, or hexadecimal numbers (octal numbers begin with 0, hexadecimal numbers begin with 0x or 0X). The default TFTP server is the client's bootpd server.

sm Specifies the host subnet mask. The *Value* is a single IP address in standard Internet dot notation. The IP address can use decimal, octal, or hexadecimal numbers (octal numbers begin with 0, hexadecimal numbers begin with 0x or 0X).

tc Specifies the table continuation, which points to a similar "template" host entry. The *Value* may be the host name or IP address of any host entry previously listed in the configuration file.

The **tc** tag is used to list commonly shared values for certain tags (such as name servers) in one host entry. Often, this template entry is a dummy host that does not actually exist and never sends the **bootpd** daemon requests. This tag may appear anywhere in the host entry.

Information explicitly specified for a host always overrides information implied by a **tc** tag, regardless of its location within the entry.

If it is necessary to delete a specific tag after it has been inferred through the **tc** tag, use the :*tag*@: construction to remove the effect of the tag feature. For example, to completely undo an IEN-116 name server specification, use :*ns*@: at an appropriate place in the configuration entry. If a tag has been deleted, it can be reset through the **tc** tag.

to Time zone offset of client's subnet in seconds from Coordinated Universal Time.

ts Time server address list.

The following tags are specific to the Xstation product:

T170 Specifies the port number used by the **x_st_mgr** daemon. The *Value* is specified in ASCII hexadecimal and must equal the port number assigned to the **x_st_mgrd** in the **/etc/services** file. If the **x_st_mgr** is on port 7000, the tag is **T170** = 1B58. If the **x_st_mgr** is on port 9000, the tag is **T170** = 2328.

T175 Specifies the host as a primary or a secondary boot server. If the *Value* is `00`, the host is a primary boot server for this Xstation. If the response *Value* is non-zero, then the host is a secondary server and will respond to a boot request for this client only after the delay time of *Value* seconds has elapsed.

In a network environment, it is desirable to have multiple hosts from which an Xstation can boot, so the Xstation can boot from an alternate server when the default host is not active. A problem can develop if a large number of Xstations continue to boot from one reliable alternate server when their default servers are back in operation. This problem is alleviated by requiring secondary servers to wait the specified number of seconds to give the primary server a chance to respond.

In addition, an Xstation will attempt to boot from the file server used for the previous boot. If the Xstation booted from the primary server, the server boots the Xstation immediately. If the Xstation booted from a secondary server, then the server will wait the specified delay time to give the primary server a chance to respond.

T176 Specifies the input device for the Xstation. If the value is `00`, the input device is the mouse. If the value is non-zero, the tablet is the input device, with value equal to the port number on the Xstation to which the tablet is connected (for example, `01` indicates that the tablet is connected to serial port 1 on the Xstation).

T177 Specifies how an Xstation 130 uses a fixed disk, if a fixed disk is present. The optional fixed disk for the Xstation 130 contains storage for a Pixmap/Font cache file and all the files needed for boot and terminal operation, if the user so specifies. When an Xstation uses the **bootp** program, the **bootp** response contains the value of the **T177** tag field.

To use the fixed-disk functions, enter a 16-bit hexadecimal tag field in the **/etc/bootptab** file entry for the Xstation 130, using the identifier **T177**.

The tag field bits are defined as follows:

Bit 0 reserved.

Bit 1 reserved.

Bits 2-4 These bits indicate how to use the fixed disk for files that are not font-related. The bit values are defined as follows:

0x00 Use the network file server for the boot file. Do not access the fixed disk.

0x01 Use the fixed disk for the boot file. Do not access the network file server.

0x02 Use the fixed disk for the boot file, but use the network file server as backup if the fixed disk cannot satisfy the request.

0x03 Use the network file server for the boot file, but use the fixed disk as backup if the network file server cannot satisfy the request.

0x04 Use the fixed disk for the boot file and if the file does not exist on the fixed disk, read the file from the network file server and store it on the fixed disk.

0x05 Use the network file server for the boot file and write the boot file to the fixed disk. This updates the boot file on the fixed disk.

Bits 5-7 These bits indicate how to use the fixed disk for font files. The bit values are defined as follows.

0x00 Use the network file server for all files. Do not access the fixed disk.

0x01 Use the fixed disk for all files. Do not access the network file server.

0x02 Use the fixed disk for all files but use the network file server as back up if the fixed disk cannot satisfy the request.

0x03 Use the network file server for all files but use the fixed disk as back up if the network file server cannot satisfy the request.

0x04 Use the fixed disk for all files, and if a file does not exist on the fixed disk, read the file from the network file server and store it on the fixed disk.

0x05 Use the fixed disk for all files. However, if the network file server has a more recent version of the file, then read the file from the server, store it on the fixed disk, and use the new file.

Bit 8 Page pixmaps to fixed disk. If set, pixmaps are paged to the fixed-disk page file. If clear, the network file server manages the page file.

Bits 9-16 reserved.

Example: **T177** = `0190` uses the fixed disk for the boot file, and all other files, and page pixmaps to the fixed disk.

T178 Specifies the usage of XDMCP on the Xstation. If the value is `00`, XDMCP mode on the Xstation is disabled. If the value is non-zero, XDMCP mode on the Xstation is as follows:

01 Broadcast

02 Direct

03 Indirect

T179 Specifies the xdmcp host used by the Xstation. The value is the host name (ASCII string) surrounded by double quotation marks.

T180 Specifies the usage of hardware pan. If the value is `00`, hardware pan is disabled. If the value is non-zero (for example,`01`), hardware pan is enabled.

T183 Specifies where the LPF keys are attached:

T183=00 LPF keys not configured

T183=01 LPF keys attached to serial port 0 (Xstation 120 or Xstation 130)

T183=02 LPF keys attached to serial port 1 (Xstation 130 only)

T183=03 LPF keys attached to serial port 2 (Xstation 130 only)

T183=04 LPF keys attached to serial port 3 (Xstation 130 only)

Guidelines and Restrictions

- Blank lines and lines beginning with # are ignored when the file is read.
- Host entries are separated from one another by new lines; a single host entry may be extended over multiple lines if the lines end with a backslash (\). However, individual host entries must not exceed 1024 characters.
- Lines in the configuration file may be longer than 80 characters.
- Tags can be displayed in any order, with the following exceptions:
  - The host name must be the first field in an entry, and
  - The hardware type must precede the hardware address.

Examples

1. The following is an example of an X-station bootptab entry with the following specifications:

```
Client’s full host name = e-jack.austin.ibm.com
host network type = 10Mb Ethernet
hardware address = 08005a7a7e84
Client’s IP address = Look up IP address using Client’s hostname
Client’s home directory = /etc/x_st_mgr
Client’s boot file (not full path name) = 130e
Port Number used by the x_st_mgr daemon = 7000
IP address of the domain name server = 9.3.199.2
IP address of the gateway = 192.100.61.1
Network subnet mask = 255.255.255.0
e-jack.austin.ibm.com:\
                :ht=ether:\
                :ha=08005a7a7e84:\
                :ip:\
                :hd=/etc/x_st_mgr:\
                :bf=130e:\
                :T170=1b58:\
                :ds=9.3.199.2:\
                :gw=192.100.61.1:\
                :sm=255.255.255.0:
```

2. The above example can also be split into two entries with one entry containing commonly used information. Then, several X-station host entries could reference this information using the **tc** tag. For instance, the following information might be common among several bootp clients.

```
host network type                       = 10Mb Ethernet
client’s home directory                 = /etc/x_st_mgr
client’s boot file (not full path name) = 130e
Port Number used by the x_st_mgr daemon = 7000
IP address of the domain name server    = 9.3.199.2
IP address of the gateway               = 192.100.61.1
Network subnet mask                     = 255.255.255.0
```

So a dummy host entry called `x_st_mgr.130e` could be created that contained this information.

```
x_st_mgr.130e:ht=ether:hd=/etc/x_st_mgr:bf=130e:T170=1b58:\
             :ds=9.3.199.2:gw=192.100.61.1:sm=255.255.255.0:
```

Then, an X-station bootptab entry could reference this information using the **tc** tag.

```
e-jack.austin.ibm.com:tc=x_st_mgr.130e:ha=08005a7a7e84:ip:
```

3. If a client uses all the information in the dummy host entry with a few exceptions, the client's host entry could specify alternate tag definitions for tags already defined in the dummy host entry. For instance, lets say that the `e-jack.austin.ibm.com` X-station uses a different domain name server than the one specified in `x_st_mgr.130e`. Then, either of the following entries could be used.

```
e-jack.austin.ibm.com:tc=x_st_mgr.130e:ds=129.35.130.95:\
                     :ha=08005a7a7e84:ip:
e-jack.austin.ibm.com:ds=129.35.130.95:tc=x_st_mgr.130e:\
                     :ha=08005a7a7e84:ip:
```

4. A client can specify a different TFTP server using the **sa** tag. For instance, if the X-station `e-jack.austin.ibm.com` gets its boot file from a server other than the bootp server, then the following entry could be used:

```
e-jack.austin.ibm.com:tc=x_st_mgr.130e:\
                     :sa="statacama.austin.ibm.com":\
                     :ha=08005a7a7e84:ip:
```

5. Tags defined in a dummy host entry referenced by a Client's host entry can be deleted using the `:tag@:` syntax. For instance, if the host entry for `e-jack.austin.ibm.com` does not need a gateway defined, the following entry could be used.

```
e-jack.austin.ibm.com:tc=x_st_mgr.130e:gw@:ha=08005a7a7e84:ip:
```

Related Information

Character Set Description (charmap) Source File Format

Purpose

Defines character symbols as character encodings.

Description

The character set description (**charmap**) source file defines character symbols as character encodings. The **/usr/lib/nls/charmap** directory contains **charmap** source files for supported locales. The **localedef** command recognizes two sections in **charmap** source files, the **CHARMAP** section and the **CHARSETID** section:

CHARMAP
: Maps symbolic character names to code points. This section must precede all other sections, and is mandatory.

CHARSETID
: Maps the code points within the code set to a character set ID. This sections is optional.

The CHARMAP Section

The **CHARMAP** section of the **charmap** file maps symbolic character names to code points. All supported code sets have the portable character set as a proper subset. Only symbols that are not defined in the portable character set must be defined in the **CHARMAP** section. The portable character set consists of the following character symbols (listed by their standardized symbolic names) and encodings:

Symbol Name	Code (hexadecimal)
`<NUL>`	000
`<SOH>>`	001
`<STX>`	002
`<ETX>`	003
`<EOT>`	004
`<ENQ>`	005
`<ACK>`	006
`<alert>`	007
`<backspace>`	008
`<tab>`	009

<new-line>	00A
<vertical-tab	00B
<form-feed>	00C
<carriage-return>	00D
<SO>	00E
<SI>	00F
<DLE>	010
<DC1>	011
<DC2>	012
<DC3>	013
<DC4>	014
<NAK>	015
<SYN>	016
<ETB>	017
<CAN>	018
<EM>	019
<SUB>	01A
<ESC>	01B
<IS4>	01C
<IS3>	01D
<IS2>	01E
<IS1>	01F
<space>	020
<exclamation-mark>	021
<quotation-mark>	022
<number-sign>	023
<dollar-sign>	024
<percent>	025
<ampersand>	026
<apostrophe>	027

<left-parenthesis>	028
<right-parenthesis>	029
<asterisk>	02A
<plus-sign>	02B
<comma>	02C
<hyphen>	02D
<period>	02E
<slash>	02F
<zero>	030
<one>	031
<two>	032
<three>	033
<four>	034
<five>	035
<six>	036
<seven>	037
<eight>	038
<nine>	039
<colon>	03A
<semi-colon>	03B
<less-than>	03C
<equal-sign>	03D
<greater-than>	03E
<question-mark>	03F
<commercial-at>	040
<A>	041
<B>	042
<C>	043
<D>	044
<E>	045

<F>	046
<G>	047
<H>	048
<I>	049
<J>	04A
<K>	04B
<L>	04C
<M>	04D
<N>	04E
<O>	04F
<P>	050
<Q>	051
<R>	052
<S>	053
<T>	054
<U>	055
<V>	056
<W>	057
<X>	058
<Y>	059
<Z>	05A
<left-bracket>	05B
<backslash>	05C
<right-bracket>	05D
<circumflex>	05E
<underscore>	05F
<grave-accent>	060
<a>	061
<b>	062
<c>	063

`<d>`	064
`<e>`	065
`<f>`	066
`<g>`	067
`<h>`	068
`<i>`	069
`<j>`	06A
`<k>`	06B
`<l>`	06C
`<m>`	06D
`<n>`	06E
`<o>`	06F
`<p>`	070
`<q>`	071
`<r>`	072
`<s>`	073
`<t>`	074
`<u>`	075
`<v>`	076
`<w>`	077
`<x>`	078
`<y>`	079
`<z>`	07A
`<left-brace>`	07B
`<vertical-line>`	07C
`<right-brace>`	07D
`<tilde>`	07E
`<DEL>`	07F

The **CHARMAP** section contains the following sections:

- The `CHARMAP` section header.
- An optional special symbolic name-declarations section. The symbolic name and value must be separated by one or more blank characters. The following are the special symbolic names and their meanings:

<code_set_name>	Specifies the name of the coded character set for which the **charmap** file is defined. This value determines the value returned by the **nl_langinfo** subroutine. The **<code_set_name>** must be specified using any character from the portable character set, except for control and space characters.
<mb_cur_max>	Specifies the maximum number of bytes in a multibyte character for the encoded character set. Valid values are 1 to 4. The default value is 1.
<mb_cur_min>	Specifies the minimum number of bytes in a multibyte character for the encoded character set. Since all supported code sets have the portable character set as a proper subset, this value must be 1.
<escape_char>	Specifies the escape character that indicates encodings in hexadecimal or octal notation. The default value is a \ (backslash).
<comment_char>	Specifies the character used to indicate a comment within a **charmap** file. The default value is a # (pound sign). With the exception of optional comments following a character symbol encoding, comments must start with a comment character in the first column of a line.

- Character set mapping statements for the defined code set.

  Each statement in this section defines a symbolic name for a character encoding. A character symbol begins with the **<** (less-than) character and ends with the **>** (greater-than) character. The characters between the **<** (less-than) and **>** (greater-than) can be any characters from the portable character set, except for control and space characters. The **>** (greater-than) character may be used if it is escaped with the escape character (as specified by the **<escape_char>** special symbolic name). A character symbol cannot exceed 32 characters in length.

  The format of a character symbol definition is:

```
<char_symbol>   encoding
      optional comment
```

  An encoding is specified as one or more character constants, with the maximum number of character constants specified by the **<mb_cur_max>** special symbolic name. The **localedef** command supports decimal, octal, and hexadecimal constants with the following formats:

```
hexadecimal constant   \xddd
octal constant         \oddd
decimal constant       \dddd
```

  Some examples of character symbol definitions are:

```
<A>            \d65             decimal constant
<B>            \x42             hexadecimal constant
<j10101>       \x81\d254        mixed hex and decimal constants
```

A range of one or more symbolic names and corresponding encoding values may also be defined, where the nonnumeric prefix for each symbolic name is common, and the numeric portion of the second symbolic name is equal to or greater than the numeric portion of the first symbolic name. In this format, a symbolic name value consists of zero or more nonnumeric characters followed by an integer of one or more decimal digits. This format defines a series of symbolic names. For example, the string <j0101>...<j0104> is interpreted as the <j0101>, <j0102>, <j0103>, and <j0104> symbolic names, in that order.

In statements defining ranges of symbolic names, the encoded value is the value for the first symbolic name in the range. Subsequent symbolic names have encoding values in increasing order. For example:

```
<j0101>...<j0104>     \d129\d254
```

This character set mapping statement is interpreted as follows:

```
<j0101>         \d129\d254
<j0102>         \d129\d255
<j0103>         \d130\d0
<j0104>         \d130\d1
```

Symbolic names must be unique, but two or more symbolic names can have the same value.

- The END CHARMAP section trailer.

Examples

The following is an example of a portion of a possible **CHARMAP** section from a **charmap** file:

```
CHARMAP
<code_set_name>         ISO8859-1
<mb_cur_max>            1
<mb_cur_min>            1
<escape_char>           \
<comment_char>          #
<NUL>                       \x00
<SOH>                       \x01
<STX>                       \x02
<ETX>                       \x03
<EOT>                       \x04
<ENQ>                       \x05
<ACK>                       \x06
<alert>                     \x07
<backspace                  \x09
<tab>                       \x09
<newline>                   \x0a
<vertical-tab>              \x0b
<form-feed>                 \x0c
<carriage-return>           \x0d
END CHARMAP
```

The CHARSETID Section

The **CHARSETID** section maps the code points within the code set to a character set ID. The **CHARSETID** section contains the following sections:

- The `CHARSETID` section header.
- Character set ID mappings for the defined code sets.
- The `END CHARSETID` section trailer.

Character set ID mappings are defined by listing symbolic names or code points for symbolic names and their associated character set IDs. The following are possible formats for a character set ID mapping statement:

```
<character_symbol>                                  number
<character_symbol>...<character_symbol>             number
character_constant                                  number
character_constant...character_constant             number
```

The `<character_symbol>` used must have previously been defined in the **CHARMAP** section. The `character_constant` must follow the format described for the **CHARMAP** section.

Individual character set mappings are accomplished by indicating either the symbolic name (defined in the **CHARMAP** section or the portable character set) followed by the character set ID, or the code point associated with a symbolic name followed by the character set ID value. Symbolic names and code points must be separated from a character set ID value by one or more blank characters. Ranges of code points can be mapped to a character set ID value by indicating appropriate combinations of symbolic names and code point values as endpoints to the range, separated by `...` (ellipsis) to indicate the intermediate characters, and followed by the character set ID for the range. The first endpoint value must be less than or equal to the second end point value.

Examples

The following is an example of a portion of a possible **CHARSETID** section from a **charmap** file:

```
CHARSETID
<space>...<nobreakspace>          0
<tilde>...<y-diaeresis>           1
END CHARSETID
```

Implementation Specifics

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

Related Information

Locale Definition Source File Format , Locale Method Source File Format .

For specific information about the locale categories and keywords, see the **LC_COLLATE** category, **LC_CTYPE** category, **LC_MESSAGES** category, **LC_MONETARY** category, **LC_NUMERIC** category, and **LC_TIME** category .

The **locale** command, **localedef** command.

For information on converting data between code sets, see Converters Overview for System Management, National Language Support Overview for System Management,

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 a saved user ID that differs from the real user ID does not produce a memory image. The same holds true for the group ID (GID) and effective group ID. The contents of a core dump are organized sequentially in the **core** file as follows:

Core header	Defines basic information about the core dump, and contains offsets that locate the remainder of the core dump information.
ldinfo structures	Defines loader information.
mstsave structures	Defines kernel thread state information. Since the faulting thread **mstsave** structure is directly saved in the core header, additional structures are saved here only for multi-threaded programs.
Default user stack	Contains a copy of the user stack at the time of the core dump.
Default data area	(Optional) Contains the user data section.
Memory mapped regions	(Optional) Contains the anonymously mapped regions.
vm_info structures	(Optional) Contains offset and size information for memory mapped regions.

By default, the user data is, anonymously mapped regions, and **vm_info** structures are not included in a core dump. This partial core dump includes the current process stack, thread stack, the thread **mstsave** structures, the 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, anonymously mapped regions, and **vm_info** structures are included in the core dump.

The format of the core header is defined by the **core_dump** structure (in the **core.h** header file), which is organized as follows:

Field Type	Field Name	Description	
char	c_signo	The number of the signal which caused the error	
char	c_flag	A bit field which describes the core dump type. The meanings of the bits are as follows:	
		FULL_CORE	core contains the data sections (0x01)
		CORE_VERSION_1	core was generated by version 4 or higher of the operating system (0x02)
		MSTS_VALID	core contains mstsave structures (0x04)
		CORE_BIGDATA	core contains big data (0x08)
		UBLOCK_VALID	core contains the u_block structure (0x10)
		USTACK_VALID	core contains the user stack (0x20)
		LE_VALID	core contains at least one module (0x40)
		CORE_TRUNC	core was truncated (0x80)
ushort	c_entries	The number of core dump modules	
struct ld_info *	c_tab	The offset to the beginning of the core table	
caddr_t	c_stack	The offset to the beginning of the user stack	
int	c_size	The size of the user stack	
struct mstsave	c_mst	A copy of the faulting mst	
struct user	c_u	A copy of the user structure	
int	c_nmsts	The number of **mstsave** structures referenced by the `c_msts` field	
struct mstsave *	c_msts	The offset to the other threads' **mstsave** structures	
int	c_datasize	The size of the data region	
caddr_t	c_data	The offset to user data	
int	c_vmregions	The number of anonymously mapped regions	
struct vm_info *	c_vmm	The offset to the start of the vm_info table	

Related Information

The **param.h** file.

The **adb** command, **dbx** command.

The **raise** subroutine, **setrlimit** subroutine, **setuid** subroutine, **sigaction** subroutine.

The Header Files Overview in *AIX Version 4.3 Files Reference* defines header files, describes how they are used, and lists several header files for which information is provided in this documentation.

core File Format (AIX Version 4.2)

Purpose

Contains an image of a 32-bit 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 a saved user ID that differs from the real user ID does not produce a memory image. The contents of a core dump are organized sequentially in the **core** file as follows:

Core header	Defines basic information about the core dump, and contains offsets which locate the remainder of the core dump information.
ldinfo structures	Defines loader information.
mstsave structures	Defines kernel thread state information. Since the faulting thread **mstsave** structure is directly saved in the core header, additional structures are saved here only for multi-threaded programs.
Default user stack	Contains a copy of the user stack at the time of the core dump.
Default data area	(Optional) Contains the user data section.
Memory mapped regions	(Optional) Contains the anonymously mapped regions.
vm_info structures	(Optional) Contains offset and size information for memory mapped regions.

The **core_dump** structure, defined by the **core.h** file, occurs at the beginning of a **core** file. The **core_dump** structure includes the following fields:

Field Type	Field Name	Description	
char	c_signo	The number of the signal that caused the error	
char	c_flag	A bit field that describes the core dump type. The meanings of the bits are as follows:	
		FULL_CORE	core contains the data sections (0x01)
		CORE_VERSION_1	core was generated by version 4 or higher of the operating system (0x02)
		MSTS_VALID	core contains mstsave structures (0x04)
		CORE_BIGDATA	core contains big data (0x08)
		UBLOCK_VALID	core contains the u_block structure (0x10)
		USTACK_VALID	core contains the user stack (0x20)
		LE_VALID	core contains at least one module (0x40)
		CORE_TRUNC	core was truncated (0x80)
ushort	c_entries	The number of core dump modules	
struct ld_info *	c_tab	The offset to the beginning of the core table	
caddr_t	c_stack	The offset to the beginning of the user stack	
int	c_size	The size of the user stack	
struct mstsave	c_mst	A copy of the faulting mst	
struct user	c_u	A copy of the user structure	
int	c_nmsts	The number of **mstsave** structures referenced by the `c_msts` field	
struct mstsave *	c_msts	The offset to the other threads' **mstsave** structures	
int	c_datasize	The size of the data region	
caddr_t	c_data	The offset to user data	
int	c_vmregions	The number of anonymously mapped regions	
struct vm_info *	c_vmm	The offset to the start of the vm_info table	

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, anonymously mapped regions, and **vm_info** structures are not included in a core dump. This partial core dump includes the current thread stack, the thread **mstsave** structures, the user structures, 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, anonymously mapped regions, and **vm_info** structures are included in the core dump.

Related Information

The **param.h** file.

The **adb** command, **dbx** command.

The **raise** subroutine, **setrlimit** subroutine, **setuid** subroutine, **sigaction** subroutine.

The Header Files Overview in *AIX Version 4.3 Files Reference* defines header files, describes how they are used, and lists several header files for which information is provided in this documentation.

core File Format (AIX Version 4.3)

Purpose

Contains an image of a 32-bit or 64-bit process at the time of an error, in the AIX 4.3 file format.

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 a saved user ID that differs from the real user ID does not produce a memory image. The contents of a core dump are organized sequentially in the **core** file as follows:

Core header	Defines basic information about the core dump, and contains offsets that locate the remainder of the core dump information.
ldinfo structures	Defines loader information.
thrdctx structures	Defines kernel thread state information. Since the faulting thread **thrdctx** structure is directly saved in the core header, additional structures are saved here only for multi-threaded programs.
segregion structures	Contains the address, size, and type for shared memory segments of the faulting process.
Default user stack	Contains a copy of the user stack at the time of the core dump.
Default data area	(Optional) Contains the user data section.
vm_infox structures	(Optional) Contains offset and size information for memory mapped regions.
Memory mapped regions	(Optional) Contains the memory mapped regions.

The **core_dumpx** structure, defined by the **core.h** file, occurs at the beginning of a **core** file. The **core_dumpx** structure includes the following fields:

Field Type	Field Name	Description	
char	c_signo	The number of the signal that caused the error	
char	c_flag	A bit field that describes the core dump type. The meanings of the bits are as follows:	
		FULL_CORE	core contains the data sections (0x01)
		CORE_VERSION_1	core was generated by version 4 or higher of the operating system (0x02)

		MSTS_VALID	core contains mstsave structures (0x04)
		CORE_BIGDATA	core contains big data (0x08)
		UBLOCK_VALID	core contains the u_block structure (0x10)
		USTACK_VALID	core contains the user stack (0x20)
		LE_VALID	core contains at least one module (0x40)
		CORE_TRUNC	core was truncated (0x80)
ushort	c_entries	The number of core dump modules	
int	c_version	Core file format version	
unsigned long long	c_fdsinfox	Offset to fd region in file	
unsigned long long	c_loader	Offset to loader region in file	
unsigned long long	c_lsize	Size of the loader region	
uint	c_n_thr	Number of elements in thread table	
unsigned long long	c_thr	Offset to thread context table	
unsigned long long	c_segs	Number of elements in segregion	
unsigned long long	c_segregion	Offset to start of segregion table	
unsigned long long	c_stack	Offset of user stack in file	
unsigned long long	c_stackorg	Base address of user stack region	
unsigned long long	c_size	Size of user stack region	
unsigned long long	c_data	Offset to user data region	
unsigned long long	c_dataorg	Base address of user data region	
unsigned long long	c_datasize	Size of user data region	
unsigned long long	c_sdorg	Base address of privately loaded region	
unsigned long long	c_sdsize	Size of user privately loaded region	
unsigned long long	c_vmregions	Number of anonymously mapped areas	
unsigned long long	c_vmm	Offset of start of vm_infox table	
struct thrdctx	c_flt	Faulting thread's context and thread data	
struct userx	c_u	Representation of the user struct and process data	

The `c_flt` field in the core dump contains the **thrdctx** structure of the faulting thread. The **thrdctx** structure includes the thread data and registers as they existed at the time of the fault. The format of the thread context structure is defined by **thrdctx** structure (in the **core.h** header file) as follows:

thrdctx	**thrdsinfo64**	thread data (in **procinfo.h** header file)
	__context64	state of registers if 64-bit process, or
	mstsave	state of registers if 32-bit process

The c_u field follows this information in the core dump. The c_u field contains the **userx** structure including the user structure fields, and the process data as they existed at the time of the fault. The format of the process information structure is defined by the **userx** structure (in the **core.h** header file) as follows:

userx	**procsinfo64**	process data (in **procinfo.h** header file)

The **ld_info** structure and then the **thrdctx** structures of the other threads (if the process is multi-threaded) follow in the core dump.

The **segregion** structure and then the user-mode stack follow in the core dump.

The **segregion** structure contains the information about a shared memory region of the faulting process.

segregion	**addr**	segment start address
	size	size of the segment
	segflags	type of the document

The first three fields of the **core_dumpx** header in AIX 4.3 are the same as that of the **core_dump** header in AIX 4.2. However, the c_entries are always zero on AIX 4.3 systems to distinguish them from the AIX 4.2 core file formats. Further, the pi_flags2 field of the **procsinfo64** structure determines if the core file is of a 32-bit process or a 64-bit process.

The AIX 4.3 system can be forced to create core files in an AIX 4.2 core file format via the SMIT tool. However, this enforcement is valid only for 32-bit processes.

By default, the user data, anonymously mapped regions, and **vm_infox** structures are not included in a core dump. This partial core dump includes the current thread stack, the thread **thrdctx** structures, the 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** or **setrlimit64** 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 user data section, **vm_infox**, and anonymously mapped region structures are included in the core dump.

Related Information

The **param.h** file.

The **adb** command, **dbx** command.

The **raise** subroutine, **setrlimit** and setrlimit64 subroutines, **setuid** subroutine, **sigaction** subroutine.

The Header Files Overview in *AIX Version 4.3 Files Reference* defines header files, describes how they are used, and lists several header files for which information is provided in this documentation.

cpio File Format

Purpose

Describes the copy in/out (**cpio**) archive file.

Description

The **cpio** utility backs up and recovers 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 reads 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, and other items describing the file, are recorded in an element of the array of structures with varying lengths.

Note: Files saved with the **-c** flag must be restored with the **-c** flag.

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`	Device that contains a directory entry for this file.
`h_ino`	I-node number that identifies the input file to the file system.
`h_mode`	Mode of the input file, as defined in the **mode.h** file.
`h_uid`	User ID of the owner of the input file.
`h_gid`	Group ID of the owner of the input file.

For remote files, these fields contain the ID after reverse translation:

`h_nlink` Number of links that are connected to the input file.

`h_rdev` ID of the remote device from which the input file is taken.

`h_mtime` Time when data was last modified. For remote files, this field contains the time at the server. This time can be changed by the **creat**, **fclearf**, **truncate**, **mknod**, **openx**, **pipe**, **utime**, or **writex** subroutine.

`h_namesize` Length of the path name, including the terminating null byte.

`h_filesize` Length of the file in bytes. This is the length of the data section that follows the header structure.

`h_name` 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)`. That is, 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.

Related Information

The **mode.h** file, **stat.h** file.

The **cpio** command, **find** command.

The **fclear** subroutine, **truncate** or **ftruncate** subroutine, **mknod** subroutine, **open**, **openx**, or **creat** subroutine, **pipe** subroutine, **scanf**, **fscanf**, **sscanf**, **wsscanf** subroutine, **utime** subroutine, **write**, **writex**, **writev**, or **writevx** subroutine.

The Header Files Overview in *AIX Version 4.3 Files Reference* defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

Devices File Format for BNU

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 **/etc/uucp/Devices** file and its augmentations and alternatives specified in the **/etc/uucp/ Sysfiles** 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 **/etc/uucp/Dialers** file.

The fields appear on the line as follows:

Type Line Line2 Class Dialer-Token Pairs

Every field of a line in the **Devices** file must contain an entry. If a field does not apply to the particular type of device or system, use a - (minus sign) 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 with 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 **/etc/uucp/Systems** file.

Line Field

The device name for the line, or port, used in the communication link is inserted here. 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 minus sign as a placeholder.

Line2 Field

Unless you are using an 801 dialer, use a **-** (minus sign) 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, use the transmission rate of the device connecting the two systems. For a telephone connection, use 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` or `1200`) 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 **/etc/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 successive speeds.

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 **-** (minus sign) 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 **/etc/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 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 **/etc/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 **/etc/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 **/etc/uucp/Systems** file and pass it to the appropriate *dialer script* (entry) in the **/etc/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 **/etc/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 **/etc/uucp/Systems** file. If the *Token* field does not represent a telephone number, enter the predefined string necessary to reach the dialer.

Examples

Setting Up Entries for Hardwired Connections

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 `-` (minus sign) is used as a placeholder in other types of connections, as 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

1. 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 `-` (minus sign) 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.
2. 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 `-` (minus sign) placeholder. The *Dialer* field part of the *Dialer-Token Pair* field is specified as a `hayes` modem, and the *Token* field is left blank.

3. 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 entered with an uppercase `A`.

4. 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

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. minus signs 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.

1. 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` the 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
```

2. 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 **/etc/uucp/Systems** files and dialing codes in their **/etc/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp directory	Contains all the configuration files for BNU, including the **Devices** file.
/etc/uucp/Dialcodes file	Contains dialing code abbreviations.
/etc/uucp/Dialers file	Specifies initial handshaking on a connection.
/etc/uucp/Systems file	Describes accessible remote systems.
/etc/uucp/Sysfiles file	Specifies possible alternative or augmentative files for **/etc/uucp/Devices**.

Related Information

The **cu** command, **uucp** command, **uucpadm** command, **uuto** command, **uux** command.

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

Dialcodes File Format for BNU

Purpose

Contains the initial digits of telephone numbers used to establish remote connections over a phone line.

Description

The **/etc/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 **/etc/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 **/etc/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 **/etc/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 **/etc/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.

Notes:

1. 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 **/etc/uucp/Systems** file rather than use dial-code abbreviations.
2. 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 **/etc/uucp/Systems** file.
3. Only someone with root user authority can edit the **Dialcodes** file, which is owned by the

uucp program login ID.

Example

The **Dialcodes** file on system `venus` contains the following dial-code prefix for use with a number in the **/etc/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp directory	Contains all the configuration files for BNU, including the **Dialcodes** file.
/etc/uucp/Devices file	Contains information about available devices.
/etc/uucp/Dialers file	Specifies initial handshaking on a connection.
/etc/uucp/Systems file	Describes accessible remote systems.
/etc/uucp/Sysfiles file	Specifies possible files used instead of **/etc/uucp/System** file, **/etc/uucp/Devices** file, and **/etc/uucp/Dialers** file.

Related Information

The **cu** command, **tip** command, **uucp** command, **uucpadm** command, **uuto** command, and **uux** command.

Dialers File Format for BNU

Purpose

Lists modems used for Basic Networking Utilities (BNU) remote communications links.

Description

The **/etc/uucp/Dialers** file and its surrogates, specified in the **/etc/uucp/Sysfiles** file, lists the modems (dialers) 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(s) contains entries for each autodialer that is included in the **/etc/uucp/Devices** file or one of its surrogate files. Surrogate file are specified in the **/etc/uucp/Sysfiles** 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 a Dialers File

Every modem (dialer) 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 a **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 **Devices** file(s). When a particular device is used to make a connection, BNU uses the *Dialer-Token Pair* field in the **Devices** file(s) to find the handshaking entry in the **Dialers** file(s).

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 a **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 `-` (minus sign) represents *pause*.
`""`	Wait for nothing; continue with the rest of the string.
`WAIT=n`	Enter this before any send string in the Dialers file, where `n` is the number of seconds to wait before timing out.

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, consists of 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 appropriate line from a **Dialers** file is interpreted to perform the dialer negotiations.

The handshaking characters include the following key sequences:

Sequence	Result
\c	Suppress new line (\n)
\D	Raw phone number
\T	Translated phone number
\N	Null character (\0)
\b	Backspace
\n	New line
\r	Carriage return
\s	Space
\t	Tab
\\	Backslash
\E	Turn echo check on
\e	Turn echo check off
\d	Delay two seconds
\p	Pause about 1/4 second
\K	Generate a break on the line
\M	Set tty setting CLOCAL on
\m	Turn tty setting CLOCAL off

For `direct` and `TCP` entries, omit this field.

Examples

Setting Up Entries in a 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\T
P  > 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 a **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 (`=,-,""`) 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 `-` (minus sign) 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 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 **Systems** file, with dial-code translation from the **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 **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

If your BNU configuration includes hardwired connections, a **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 a **Devices** file (see the /etc/uucp/Devices file).

Setting Up the TCP/IP Entry

If your BNU configuration includes TCP/IP connections, the **Dialers** file used by the **uucico** service 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 **uucico** service **Devices** file(s).

Setting Up Entries for Both Local and Remote Systems

The following example illustrates the entries needed in the **Dialers** file to correspond to entries in the **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 **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 **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
  ```

  Note: The **Dialers** file and **Devices** file for the system `venus` and `merlin` can be files other than **/etc/uucp/Dialers** and **/etc/uucp/Devices**. Use of the **/etc/uucp/Sysfiles** file enables a system administrator to allow the use of one or more files on each system to replace or augment the **/etc/uucp/Dialers** and **/etc/uucp/Devices** file. See the **Sysfiles Files Format for BNU** in *AIX Version 4.3 Files Reference*.

Troubleshooting Connection Problems

> **Note:** The **Dialer** and **Systems** files discussed in the section can be files other than **/etc/uucp/Dialers** and **/etc/uucp/Systems**. See the **Sysfiles Files Format for BNU** in *AIX Version 4.3 Files Reference*.

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 **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/sbin/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 **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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp directory	Contains all the configuration files for BNU, including the **Dialers** file.
/etc/uucp/Devices file	Contains information about available devices.
/etc/uucp/Dialcodes file	Contains dialing code abbreviations.
/etc/uucp/Systems file	Describes accessible remote systems.
/etc/uucp/Sysfiles file	Specifies possible alternative files for **/etc/uucp/System**, **/etc/uucp/Dialers**, and **/etc/uucp/Devices**.

Related Information

The **ct** command, **cu** command, **uukick** command, **uutry** command, **Uutry** command.

The **uucico** daemon.

Configuring BNU, Monitoring a BNU Remote Connection, Debugging BNU Login Failures Using the uucico Daemon, BNU File and

Dialing Directory File Format for ATE

Purpose

Lists phone numbers used to establish modem connections.

Description

The ATE dialing directory file lists phone numbers that the Asynchronous Terminal Emulation (ATE) uses to establish remote connections by modem.

Users name the dialing directory file with any valid file name and place it in any directory where read and write access is owned. Edit the dialing directory file with any ASCII text editor. The default dialing directory file is the **/usr/lib/dir** file.

The **connect** and **directory** subcommands of ATE access the dialing directory file. Use the **connect** command to use numbers that are not in the dialing directory file. Use the **directory** subcommand to view the dialing directory.

Users can have more than one dialing directory. To change the dialing directory file the ATE program uses, modify the **ate.def** file in the current 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 file is similar to a page in a telephone book. This file contains entries for the remote systems called 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 each entry easier to read. The fields are:

Name 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 up to 40 characters. Consult the modem documentation for a list of acceptable digits and characters. For example, if a 9 must be dialed 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 in bits per second (bps). Determines the number of characters transmitted per second. Select a baud rate that is compatible with the communication line being used. The following are acceptable rates: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19,200.

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 (off) or 1 (on).

Linefeed Adds a line-feed character at the end of each line of data coming in from a remote system. The line-feed character is similar in function to the carriage-return and new-line characters. The entry for the `Linefeed` field can be 0 (off) or 1 (on).

Examples

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` is the *Echo*, and the second `0` is the *Linefeed*.

Implementation Specifics

This file is part of Asynchronous Terminal Emulation (ATE) in BOS Extensions 2.

Files

ate.def Contains ATE default values.

/usr/lib/dir Contains the default dialing directory listing.

Related Information

The **ate** command.

The **connect** subcommand, **directory** subcommand.

DOMAIN Cache File Format for TCP/IP

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.

Examples

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: `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 Base Operating System (BOS) Runtime.

Files

/etc/named.conf	Defines how the **named** daemon initializes the DOMAIN name server file.
/usr/samples/tcpip/named.conf	Sample **named.conf** file, which also contains directions for its use.
/usr/samples/tcpip/named.data	Sample **named.data** file, which also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Data file format, DOMAIN Reverse Data file format, DOMAIN Local file format.

DOMAIN Data File Format for TCP/IP

Purpose

Stores name resolution information for the **named** daemon.

Description

The host's 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 host's 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 host's 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/samples/tcpip** directory to assist you in converting your existing **/etc/hosts** file to DOMAIN data files. The **awk** scripts 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 host's 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:** Host `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    System ABC 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    System ABC 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    System ABC 3.1
lopnor             IN       A        160.9.201.5
lopnor-xyz         IN       CNAME    lopnor.xyz.
                   IN       HINFO    System ABC 3.1
sandy              IN       A        160.9.201.3
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/etc/named.conf	Defines how the **named** daemon initializes the DOMAIN name server file.
/usr/samples/tcpip/addrs.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.rev** file. The awk script also contains directions for its use.
/usr/samples/tcpip/hosts.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.data** file. The awk script also contains directions for its use.
/usr/samples/tcpip/named.conf	Sample **named.conf** file, which also contains directions for its use.
/usr/samples/tcpip/named.data	Sample **named.data** file, which also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Reverse Data file format, DOMAIN Cache file format, DOMAIN Local file format.

Standard Resource Record Format for TCP/IP.

DOMAIN Local Data File Format for TCP/IP

Purpose

Defines the local loopback information for the **named** daemon 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 the 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: Host `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 Base Operating System (BOS) Runtime.

Files

/etc/named.conf	Defines how the **named** daemon initializes the DOMAIN name-server file.
/usr/samples/tcpip/named.conf	Sample **named.conf** file, which also contains directions for its use.
/usr/samples/tcpip/named.data	Sample **named.data** file, which also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Data file format, DOMAIN Reverse Data file format, DOMAIN Cache file format.

DOMAIN Reverse Data File Format for TCP/IP

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/samples/tcpip** directory to assist you in converting your existing **/etc/hosts** file to **named** data files. The **awk** scripts 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: Host `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
4.201                                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 Base Operating System (BOS) Runtime.

Files

/etc/named.conf	Defines how the **named** daemon initializes the DOMAIN name server file.
/usr/samples/tcpip/addrs.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.rev** file. The **awk** script also contains directions for its use.
/usr/samples/tcpip/hosts.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.data** file. The **awk** script also contains directions for its use.
/usr/samples/tcpip/named.conf	Contains a sample **named.conf** file, which also contains directions for its use.
/usr/samples/tcpip/named.data	Contains a sample **named.data** file, which also contains directions for its use.

Related Information

The **named** daemon.

The DOMAIN Data file format, DOMAIN Cache file format, DOMAIN Local Data file format.

Standard Resource Record Format for TCP/IP.

eqnchar File Format

Purpose

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

Description

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

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

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

eqn /usr/share/lib/pub/eqnchar [*Flag...*] [**--**] [*File...*] **|** **troff** [*Flag...*]

eqn /usr/share/lib/pub/cateqnchar [*Flag...*] [**--**] [*File...*] **|** **troff** [*Flag...*]

neqn /usr/share/lib/pub/eqnchar [*Flag...*] [**--**] [*File...*] **|** **nroff** [*Flag...*]

Implementation Specifics

This file is part of Formatting Tools in the Text Formatting System.

Files

/usr/share/lib/pub/cateqnchar	Contains the character definitions for **troff-**supported device.

Related Information

ftpusers File Format for TCP/IP

Purpose

Specifies local user names that cannot be used by remote FTP clients.

Description

The **/etc/ftpusers** file contains a list of local user names that the **ftpd** server does *not* allow remote File Transfer Protocol (FTP) clients to use. The format of the **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 the **ruser** command.

Examples

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 Base Operating System (BOS) Runtime.

Files

/etc/passwd	Contains user authentication information.

Related Information

The **ruser** command.

The **ftpd** daemon.

gated.conf File Format for TCP/IP

Purpose

Contains configuration information for the **gated** daemon.

Description

The **/etc/gated.conf** file contains configuration information for the **gated** daemon. The file contains a sequence of statements. Statements are composed of tokens separated by white space. You can create white space using any combination of blanks, tabs, and new lines. The **gated.conf** file supports several statements:

%directory (directive)	Sets the directory for include files.
%include (directive)	Includes a file into **gated.conf**.
traceoptions (trace)	Specifies which events are traced.
options (definition)	Defines **gated.conf** options.
interfaces (definition)	Defines **gated.conf** interfaces.
autonomoussystem	Defines the AS number.
routerid (definition)	Defines the originating router (BGP, OSPF).
martians (definition)	Defines invalid destination addresses.
rip (protocol)	Enables RIP protocol.
hello (protocol)	Enables HELLO protocol.
isis (protocol)	Enables ISIS protocol.
ospf (protocol)	Enables OSPF protocol.
EGP (protocol)	Enables EGP protocol.
bgp (protocol)	Enables BGP protocol.
icmp (protocol)	Configures the processing of general ICMP packets.
snmp (protocol)	Enables reporting to SNMP.
static (static)	Defines static routes.
import (control)	Defines which routes to import.
export (control)	Defines which routes to export.
aggregate (control)	Defines which routes to aggregate.
generate (control)	Defines which routes to generate.

Directive Statements

Directive statements provide direction to the **gated.conf** configuration language parser about included files and the directories in which these files reside. Directive statements are immediately acted upon by the parser. Other statements terminate with a semi-colon (;), but directive statements terminate with a newline. The two directive statements are:

%directory "*directory*" Defines the directory where the include files are stored. When it is used, **gated.conf** looks in the directory identified by pathname for any included files that do not have a fully qualified filename, that is, do not begin with "/". This statement does not actually change the current directory, it just specifies the prefix applied to included file names.

%include "*filename*" Identifies an include file. The contents of the file are *included* in the **gated.conf** file at the point in the **gated.conf** file where the `%include` directive is encountered. If the filename is not fully qualified, that is, does not begin with "/", it is considered to be relative to the directory defined in the `%directory` directive. The `%include` directive statement causes the specified file to be parsed completely before resuming with this file. Nesting up to ten levels is supported.

In a complex environment, segmenting a large configuration into smaller more easily understood segments might be helpful, but one of the great advantages of **gated.conf** is that it combines the configuration of several different routing protocols into a single file. Segmenting a small file unnecessarily complicates routing configurations.

Trace Statements

Trace statements control tracing options. **gated.conf**'s tracing options may be configured at many levels. Tracing options include the file specifications, control options, and global and protocol specific tracing options. Unless overridden, tracing options from the next higher level are inherited by lower levels. For example, BGP peer tracing options are inherited from BGP group tracing options, which are inherited from global BGP tracing options, which are inherited from global **gated.conf** tracing options. At each level, tracing specifications override the inherited options.

Global tracing options

There are two types of global options, those that only affect global operations, and those that have potential significance to protocols.

Global significance only

The trace flags that only have global significance are:

parse Traces the lexical analyzer and parser. Mostly used by **gated.conf** developers for debugging.

adv Traces the allocation of and freeing of policy blocks. Mostly used by the **gated.conf** developers for debugging.

symbols Used to trace symbols read from the kernel at startup. The only useful way to specify this level of tracing is via the **-t** option on the command line since the symbols are read from the kernel before parsing the configuration file.

iflist Used to trace the reading of the kernel interface list. It is useful to specify this with the **-t** option on the command line since the first interface scan is done before reading the configuration file.

Protocol significance

The options flags that have potential significance to protocols are:

all Turn on all of the following.

general Shorthand notation for specifying both normal and route.

state Trace state machine transitions in the protocols.

normal Trace normal protocols occurrences. Abnormal protocol occurrences are always traced.

policy Trace application of protocol and user-specified policy to routes being imported and exported.

task Trace system interface and processing associated with this protocol or peer.

timer Trace timer usage by this protocol or peer.

route Trace routing table changes for routes installed by this protocol or peer.

Note: Not all of the above options apply to all of the protocols. In some cases, their use does not make sense (for instance, RIP does not have a state machine) and in some instances the requested tracing has not been implemented (such as RIP support of the policy option).

Note: It is not currently possible to specify packet tracing from the command line. This is because a global option for packet tracing would potentially create too much output.

When protocols inherit their tracing options from the global tracing options, tracing levels that don’t make sense (such as parse, adv and packet tracing options) are masked out.

Global tracing statements have an immediate effect, especially parsing options that affect the parsing of the configuration file. Tracing values inherited by protocols specified in the configuration file are initially inherited from the global options in effect as they are parsed, unless they are overridden by more specific options. After the configuration file is read, tracing options that were not explicitly specified are inherited from the global options in effect at the end of the configuration file.

Packet tracing

Tracing of packets is very flexible. For any given protocol, there are one or more options for tracing packets. All protocols allow use of the **packets** keyword that allows for tracing all packets sent and received by the protocol. Most protocols have other options for limiting tracing to a useful subset of packet types. These tracing options can be further controlled with the following modifiers:

detail

The `detail` must be specified before `send` or `recv`. Normally packets are traced in a terse form of one or two lines. When `detail` is specified, a more verbose format is used to provide further detail on the contents of the packet.

send
recv

These options limit the tracing to packets sent or received. Without these options both sent and received packets will be traced.

Note: `Detail`, if specified, must be before `send` or `recv`. If a protocol allows for several different types of packet tracing, modifiers may be applied to each individual type. But be aware that within one tracing specification the trace flags are summed up, so specifying `detail packets` will turn on full tracing for all packets.

Traceoptions syntax

```
traceoptions ["trace_file" [replace] [size size[k|m] files files]]
        [control_options] trace_options [except trace_options] ;

    traceoptions none ;
```

trace_file	Specifies the file to receive tracing information. If this file name does not begin with a slash (/), the directory where **gated** was started is prepended to the name.
replace	Indicates tracing should start by replacing an existing file. The default is to append to an existing file.
size *size*[**k**|**m**] **files** *files*	Limits the maximum size of the trace file to the specified size (minimum 10k). When the trace file reaches the specified size, it is renamed to `file.0`, then `file.1`, `file.2` up to the maximum number of files (minimum specification is 2).
control_options	Specifies options that control the appearance of tracing. Valid values are: **nostamp** — Specifies that a timestamp should not be prepended to all trace lines.
except *trace_options*	Used to enable a broad class of tracing and then disable more specific options.
none	Specifies that all tracing should be turned off for this protocol or peer.

Options Statements

Options statements allow specification of some global options. If used, `options` must appear before any other type of configuration statement in the **gated.conf** file.

The options statement syntax is:

```
options
    [nosend ]
    [noresolv ]
    [gendefault [preference preference ] [gateway gateway] ]
    [syslog [upto ] log_level ]
    [mark time ]
    ;
```

The options list can contain one or more of the following options:

gendefault [**preference** *preference*] [**gateway** *gateway*]	When `gendefault` is enabled when a BGP or EGP neighbor is up, it causes the creation of a default route with the special protocol `default`. This can be disabled per BGP/EGP group with the `nogendefault` option. By default, this route has a preference of 20. This route is normally not installed in the kernel forwarding table, it is only present so it can be announced to other protocols. If a gateway is specified, the default route will be installed in the kernel forwarding table with a next hop of the listed gateway. **Note:** The use of the more general `generate default` option is preferred to the use of this `gendefault` option. See the section on Route Aggregation for more information on the `generate` statement.
nosend	Do not send any packets. This option makes it possible to run **gated.conf** on a live network to test protocol interactions without actually participating in the routing protocols. The packet traces in the **gated.conf** log can be examined to verify that **gated.conf** is functioning properly. This is most useful for RIP and HELLO.
noresolv	By default, **gated.conf** will try to resolve symbolic names into IP addresses; this option will prevent that.
syslog [**upto**] *log_level*	Controls the amount of data **gated.conf** logs via syslog.
mark *time*	Specifying this option causes **gated** to output a message to the trace log at the specified interval. This can be used as one method of determining if **gated** is still running.

Interface Statement

Interface Syntax

```
interfaces   {
    options
       [ strictinterfaces ]
       [ scaninterval   time ]
    ;
    interface  interface_list
       [ preference  preference ]
       [ down preference preference ]
       [ passive ]
       [ simplex ]
       [ reject ]
       [ blackhole ]
    ;
    define   address
       [ broadcast  address ] | [ pointtopoint  address ]
       [ netmask  mask ]
       [ multicast ]
    ;
} ;
```

An interface is the connection between a router and one of its attached networks. A physical interface may be specified by interface name, by IP address, or by domain name, (unless the network is an unnumbered point-to-point network.) Multiple levels of reference in the configuration language allow identification of interfaces using wildcard, interface type name, or delete word addresses. The *interface_list* is a list of one or more interface names including wildcard names (names without a number) and names that may specify more than one interface or address, or the token `all` for all interfaces.

options Allows configuration of some global options related to interfaces. These are:

strictinterfaces Indicates that it is a fatal error to reference an interface in the configuration file that is not present when the **gated** daemon is started and is not listed in a `define` statement. Without this option a warning message will be issued, but the **gated** daemon will continue.

scaninterval *time* Specifies how often **gated** daemon scans the kernel interface list for changes. Note that **gated** daemon will also scan the interface list on receipt of a `SIGUSR2`.

interface *interface_list*	Sets interface options on the specified interfaces. An interface list is `all` or a list of interface names domain names, or numeric addresses. Options available on this statement are:

preference *preference*	Sets the preference for routes to this interface when it is up and appears to be functioning properly. The default preference is `0`.
down preference *preference*	Sets the preference for routes to this interface when the **gated** daemon does not believe it to be functioning properly, but the kernel does not indicate it is down. The default value is `120`.
passive	Prevents the **gated** daemon from changing the preference of the route to this interface if it is not believed to be functioning properly due to lack of received routing information. The **gated** daemon will only perform this check if the interface is actively participating in a routing protocol.

define *address*

Defines interfaces that might not be present when the **gated** daemon is started so they may be referenced in the configuration file when `strictinterfaces` is defined. Possible `define` keywords are:

broadcast *address*
: Defines the interface as broadcast capable (for example, Ethernet or Token Ring) and specifies the broadcast address.

pointopoint *address*
: Defines the interface as a pointopoint interface (for example, SLIP or PPP) and specifies the address on the local side. The first *address* on the `define` statement references the address of the host on the **remote** end of the interface, the *address* specified after this `pointopoint` keyword defines the address on the **local** side of the interface.

  An interface not defined as broadcast or pointopoint is assumed to be non-broadcast multiaccess (NBMA), such as an X.25 network.

netmask *mask*
: Specifies the subnetmask to be used on this interface. This is ignored on pointopoint interfaces.

multicast
: Specifies that the interface is multicast capable.

Interface Lists

An interface list is a list of references to interfaces or groups of interfaces. There are four methods available for referring to interfaces. They are listed here from most general to most specific.

all	This refers to all available interfaces.
Interface name wildcard	This refers to all the interfaces of the same type. AIX interfaces consist of the name of the device driver, like `en`, and a unit number, like `0` or `5`. References to the name contain only alphabetic characters and match any interfaces that have the same alphabetic part. For example, `en` would refer to all Ethernet interfaces.
Interface name	This refers to a specific interface, usually one physical interface. These are specified as an alphabetic part followed by a numeric part. This will match one specific interface. For example, `en1` will match an interface named `en1`, but not an interface named `en10`.
Interface address	This matches one specific interface. The reference can be by protocol address (that is, `10.0.0.51`), or by symbolic hostname (that is, `hornet.ibm.com`). Note that a symbolic hostname reference is only valid when it resolves to only one address. Use of symbolic hostnames is not recommended.

If many interface lists are present in the config file with more than one parameter, these parameters are collected at run-time to create the specific parameter list for a given interface. If the same parameter is specified on more than one list, the parameter with the most specific interface is used.

For example, consider a system with three interfaces: `en0`, `en1`, and `tr0`.

```
rip yes {
    interface all noripin noripout;
    interface en ripin;
    interface en1 ripout;
} ;
```

RIP packets would only be accepted from interfaces `en0` and `en1`, but not from `tr0`. RIP packets would only be sent on interface `en1`.

IP Interface Addresses and Routes

loopback — This interface must have the address of **127.0.0.1**. Packets sent to this interface are sent back to the originator. This interface is also used as a catch-all interface for implementing other features, such as `reject` and `blackhole` routes. Although a netmask is reported on this interface, it is ignored. It is useful to assign an additional address to this interface that is the same as the OSPF or BGP `router id`; this allows routing to a system based on the `router id` that will work if some interfaces are down.

broadcast — This is a multi-access interface capable of a physical level broadcast, such as `Ethernet`, `Token Ring`, and `FDDI`. This interface has an associated subnet mask and broadcast address. The interface route to a `broadcast` network will be a route to the complete subnet.

point-to-point — This is a `tunnel` to another host, usually on some sort of `serial` link. This interface has a `local` address, and a `remote` address.

The `remote` address must be unique among all the interface addresses on a given router. The `local` address may be shared among many `point-to-point` and up to one non-`point-to-point` interface. This is technically a form of the `router id` method for addressless links. This technique conserves subnets as none are required when using this technique.

If a subnet mask is specified on a `point-to-point` interface, it is only used by RIP version 1 and HELLO to determine which subnets may be propagated to the router on the other side of this interface.

non-broadcast multi-access or **nbma** — This type of interface is multi-access, but not capable of broadcast. An example would be `frame relay` and `X.25`. This type of interface has a local address and a subnet mask.

The **gated** daemon insures that there is a route available to each IP interface that is configured and up. Normally this is done by the **ifconfig** command that configures the interface; the **gated** daemon does it to insure consistency.

For `point-to-point` interfaces, the **gated** daemon installs some special routes. If the `local` address on one or more `point-to-point` interfaces is not shared with a non-`point-to-point` interface, the **gated** daemon installs a route to the `local` address pointing at the `loopback` interface with a preference of 110. This insures that packets originating on this host destined for this `local` address are handled locally. OSPF prefers to route packets for the `local` interface across the `point-to-point` link where they will be returned by the router on the remote end. This is used to verify operation of the link. Since OSPF installs routes with a preference of 10, these routes will override the route installed with a preference of 110.

If the `local` address of one or more `point-to-point` interfaces is shared with a non-`point-to-point` interface, the **gated** daemon installs a route to the `local` with a preference of 0 that will not be installed in the forwarding table. This is to prevent protocols like OSPF from routing packets to this address across a serial interface when this system could be functioning as a `host`.

When the status of an interface changes, the **gated** daemon notifies all the protocols, which take the appropriate action. The **gated** daemon assumes that interfaces that are not marked `UP` do not exist.

The **gated** daemon ignores any interfaces that have invalid data for the `local`, `remote`, or `broadcast` addresses or the `subnet mask`. Invalid data includes zeros in any field. The **gated** daemon will also ignore any `point-to-point` interface that has the same local and remote addresses.

Definition Statements

Definition statements are general configuration statements that relate to all of **gated** daemon or at least to more than one protocol. The three definition statements are `autonomoussystem`, `routerid`, and `martians`. If used, `autonomoussystem`, `routerid`, and `martians` must appear before any other type of configuration statement in the **gated** daemon file.

Autonomous System Configuration

```
autonomoussystem autonomous_system [loops number] ;
```

Sets the autonomous system number of this router to be `autonomous system`. This option is required if BGP or EGP are in use. The AS number is assigned by the Network Information Center (NIC).

`Loops` is only for protocols supporting AS paths, such as BGP. It controls the number of times this autonomous system may appear in an AS path and defaults to 1 (one).

Router ID Configuration

```
routerid host ;
```

Sets the router identifier for use by the BGP and OSPF protocols. The default is the address of the first interface encountered by the **gated** daemon. The address of a non-point-to-point interface is preferred over the local address of a point-to-point interface and an address on a loopback interface that is not the loopback address (127.0.0.1) is most preferred.

Martian Configuration

```
    martians {
        host host [allow] ;
        network [allow] ;
        network mask mask [allow] ;
        network masklen number [allow] ;
        default [allow] ;
    } ;
```

Defines a list of `martian` addresses about which all routing information is ignored. Sometimes a misconfigured system sends out obviously invalid destination addresses. These invalid addresses, called martians, are rejected by the routing software. This command allows additions to the list of martian addresses. See the section on Route Filtering for more information on specifying ranges. Also, the *allow* parameter may be specified to explicitly allow a subset of a range that was disallowed.

Sample Definition Statements

```
options gendefault ;
autonomoussystem 249 ;
interface 128.66.12.2 passive ;
martians {
    0.0.0.26
};
```

The statements in the sample perform the following functions:

- The options statement tells the system to generate a default route when it peers with an EGP or BGP neighbor.
- The autonomoussystem statement tells the **gated** daemon to use the AS number 249 for EGP and BGP.
- The interface statement tells the **gated** daemon not to mark interface 128.66.12.2 as down even if it sees no traffic.
- The martians statement prevents routes to 0.0.0.26 from ever being accepted.

The RIP Statement

```
rip yes | no | on | off [{
    broadcast ;
    nobroadcast ;
    nocheckzero ;
    preference preference ;
    defaultmetric metric ;
    query authentication [none | [[simple|md5] password]] ;
    interface interface_list
        [noripin] | [ripin]
        [noripout] | [ripout]
        [metricin metric]
        [metricout metric]
        [version 1]|[version 2 [multicast|broadcast]]
        [[secondary] authentication [none | [[simple|md5] password]] ;
    trustedgateways gateway_list ;
    sourcegateways gateway_list ;
    traceoptions trace_options ;
} ] ;
```

The `rip` statement enables or disables RIP. If the `rip` statement is not specified, the default is `rip on ;`. If enabled, RIP will assume `nobroadcast` when there is only one interface and `broadcast` when there is more than one.

The options are as follows:

broadcast	Specifies that RIP packets will be broadcast regardless of the number of interfaces present. This is useful when propagating static routes or routes learned from another protocol into RIP. In some cases, the use of `broadcast` when only one network interface is present can cause data packets to traverse a single network twice.
nobroadcast	Specifies that RIP packets will not be broadcast on attached interfaces, even if there is more than one. If a `sourcegateways` clause is present, routes will still be unicast directly to that gateway.
nocheckzero	Specifies that RIP should not make sure that reserved fields in incoming version 1 RIP packets are zero. Normally RIP will reject packets where the reserved fields are zero.
preference *preference*	Sets the preference for routes learned from RIP. The default preference is 100. This preference may be overridden by a preference specified in import policy.
defaultmetric *metric*	Defines the metric used when advertising routes via RIP were learned from other protocols. If not specified, the default value is 16 (unreachable). This choice of values requires you to explicitly specify a metric in order to export routes from other protocols into RIP. This metric may be overridden by a metric specified in export policy.
query authentication [**none** | [[**simple**|**md5**] *password*]] **;**	Specifies the authentication required of query packets that do not originate from routers. The default is `none`.

interface *interface_list*

Controls various attributes of sending RIP on specific interfaces. See the section on interface list specification for a description of the *interface_list*.

> **Note:**If there are multiple interfaces configured on the same subnet, RIP updates will only be sent from the first one from which RIP output is configured.

The possible parameters are:

noripin
: Specifies that RIP packets received via the specified interface will be ignored. The default is to listen to RIP packets on all non-loopback interfaces.

ripin
: This is the default. This argument may be necessary when `noripin` is used on a wildcard interface descriptor.

noripout
: Specifies that no RIP packets will be sent on the specified interfaces. The default is to send RIP on all broadcast and non-broadcast interfaces when in `broadcast` mode. The sending of RIP on point-to-point interfaces must be manually configured.

ripout
: This is the default. This argument is necessary when it is desired to send RIP on point-to-point interfaces and may be necessary when `noripin` is used on a wildcard interface descriptor.

metricin *metric*
: Specifies the RIP metric to add to incoming routes before they are installed in the routing table. The default is the kernel interface metric plus 1 (which is the default RIP hop count). If this value is specified it will be used as the absolute value, the kernel metric will not be added. This option is used to make this router prefer RIP routes learned via the specified interface(s) less than RIP routes from other interfaces.

metricout *metric*
: Specifies the RIP metric to be added to routes that are sent via the specified interface(s). The default is zero. This option is used to make other routers prefer other sources of RIP routes over this router.

version 1
: Specifies that RIP packets sent via the specified interface(s) will be version 1 packets. This is the default.

version 2
: Specifies that RIP version 2 packets will be sent on the specified interfaces(s). If IP multicast support is available on this interface, the default is to send full version 2 packets. If it is not available, version 1 compatible version 2 packets will be sent.

multicast
: Specifies that RIP version 2 packets should be multicast on this interface. This is the default.

broadcast
: Specifies that RIP version 1 compatible version 2 packets should be broadcast on this interface, even if IP multicast is available.

[**secondary**] **authentication** [**none** | [**simple**|**md5**] *password*]
: This defines the authentication type to use. It applies only to RIP version 2 and is ignored for RIP-1 packets. The default authentication type is `none`. If a password is specified, the authentication type defaults to `simple`. The password should be a quoted string with between 0 and 16 characters.

: If `secondary` is specified, this defines the secondary authentication. If omitted, the primary authentication is specified. The default is primary authentication of `none` and no secondary authentication.

trustedgateways *gateway_list*	Defines the list of gateways from which RIP will accept updates. The *gateway_list* is simply a list of host names or IP addresses. By default, all routers on the shared network are trusted to supply routing information. But if the `trustedgateways` clause is specified, only updates from the gateways in the list are accepted.
sourcegateways *gateway_list*	Defines a list of routers to which RIP sends packets directly, not through multicast or broadcast. This can be used to send different routing information to specific gateways. Updates to gateways in this list are not affected by `noripout` on the interface.
traceoptions *trace_options*	Specifies the tracing options for RIP. (See Trace Statements and the RIP specific tracing options below.)

Tracing options

The `policy` option logs info whenever a new route is announced, the metric being announced changes, or a route goes or leaves holddown.

Packet tracing options (which may be modified with `detail`, `send`, or `recv`):

packets All RIP packets.

request All RIP information request packets, such as `REQUEST`, `POLL`, and `POLLENTRY`.

response All RIP `RESPONSE` packets, which are the types of packets that actually contains routing information.

other Any other type of packet. The only valid ones are `TRACE_ON` and `TRACE_OFF` both of which are ignored.

The Hello Statement

```
hello yes | no | on | off [{
    broadcast ;
    nobroadcast ;
    preference preference ;
    defaultmetric metric ;
    interface interface_list
            [nohelloin] | [helloin]
            [nohelloout] | [helloout]
            [metricin metric]
            [metricout metric] ;
    trustedgateways gateway_list ;
    sourcegateways gateway_list ;
    traceoptions trace_options ;
} ] ;
```

The `hello` statement enables or disables HELLO. If the `hello` statement is not specified, the default is `hello off`. If enabled, HELLO will assume `nobroadcast` when there is only one interface and `broadcast` when there is more than one interface.

broadcast

Specifies that HELLO packets will be broadcast regardless of the number of interfaces present. This is useful when propagating static routes or routes learned from anther protocol into HELLO. In some cases, the use of `broadcast` when only one network interface is present can cause data packets to traverse a single network twice.

nobroadcast

Specifies that HELLO packets will not be broadcast on attached interfaces, even if there are more than one. If a `sourcegateways` clause is present, routes will still be unicast directly to that gateway.

preference *preference*

Sets the preference for routes learned from HELLO. The default preference is op. This preference may be overridden by a preference specified in import policy.

defaultmetric *metric*

Defines the metric used when advertising routes via HELLO were learned from other protocols. If not specified, the default value is 30000 (unreachable). This choice of values requires you to explicitly specify a metric in order to export routes from other protocols into HELLO. This metric may be overridden by a metric specified in export policy.

interface *interface_list*

Controls various attributes of sending HELLO on specific interfaces. See the section on interface list specification for the description of the *interface_list*.

> **Note:** If there are multiple interfaces configured on the same subnet, HELLO updates will only be sent from the first one from which the HELLO output is configured.

The possible parameters are:

nohelloin
: Specifies that HELLO packets received via the specified interface will be ignored. The default is to listen to HELLO on all non-loopback interfaces.

helloin
: This is the default. This argument may be necessary when `nohelloin` is used on a wildcard interface descriptor.

nohelloout
: Specifies that no HELLO packets will be sent on the specified interfaces. The default is to send HELLO on all broadcast and non-broadcast interfaces when in `broadcast` mode. The sending of HELLO on point-to-point interfaces must be manually configured.

helloout
: This is the default. This argument is necessary when it is desired to send HELLO on point-to-point interfaces and may be necessary when `nohelloin` is used on a wildcard interface descriptor.

metricin *metric*
: Specifies the HELLO metric to add to incoming routes before they are installed in the routing table. The default is the kernel interface metric plus 1 (which is the default HELLO hop count). If this value is specified it will be used as the absolute value; the kernel metric will not be added. This option is used to make this router prefer HELLO routes learned via the specified interface(s) less than HELLO routes from other interfaces.

metricout *metric*
: Specifies the HELLO metric to be added to routes that are sent via the specified interface(s). The default is zero. This option is used to make other routers prefer other sources of HELLO routes over this router.

trustedgateways *gateway_list*

Defines the list of gateways from which HELLO will accept updates. The *gateway_list* is simply a list of host names or IP addresses. By default, all routers on the shared network are trusted to supply routing information. But if the `trustedgateways` clause is specified, only updates from the gateways in the list are accepted.

sourcegateways *gateway_list*

Defines a list of routers to which HELLO sends packets directly, not through multicast or broadcast. This can be used to send different routing information to specific gateways. Updates to gateways in this list are not affected by `noripout` on the interface.

traceoptions *trace_options*

Specifies the tracing options for HELLO. (See Trace Statements and the HELLO specific tracing options below.)

The default preference is 90. The default metric is 30000.

Tracing options

The `policy` option logs info whenever a new route is announced, the metric being announced changes, or a route goes or leaves holddown.

Packet tracing options (which may be modified with `detail`, `send`, and/or `recv`):

packets All HELLO packets

The IS-IS Statement

```
isis no | dual | ip | iso {
    level 1|2 ;
    [traceoptions <isis_traceoptions> ;]
    [systemid <6_digit_hexstring> ;]
    [area <hexstring> ;]
    [set <isis_parm> <value> ; ... ]
    circuit <string>
        metric [level 1|2] <1..63>
        ...
        priority [level 1|2] <0..127>
        ...
        ;
    ...
} ;
```

This statement enables the IS-IS protocol in the **gated** daemon. By default, IS-IS is disabled. The `dual` option specifies that the IS-IS protocol is enabled for both ISO and IP addressing. The `isis` statement consists of an initial description of the IS and a list of statements that determine the configuration of the specific circuits and networks to be managed. Statements may appear in any order and include:

level Indicates whether **gated** is running on a Level 1 (intra-area) or Level 2 (inter-area) IS. The default is Level 1.

traceoptions Covered in the Tracing options section below.

systemid Overrides the autoconfigured system ID (determined from interface addresses and corresponding netmasks). If no system identifier is specified, the system ID portion of the first real circuit's NSAP is used. Once a system ID is set, it cannot be changed without disabling and reenabling all of IS-IS.

area IS-IS area addresses are automatically configured based on the real circuits over which IS-IS runs. Addresses specified in this statement are maintained in addition to those configured automatically from the circuits. This command is used primarily for simulation.

circuit Each `circuit` statement specifies one of the circuits the system will manage. Circuits normally correspond to UNIX interfaces, with `string` being the interface name, but simulated device names may also be specified. If the string is in the form of "simN", where N is an integer, the circuit is assumed to be a simulated circuit managed by the network simulator troll. The circuit attributes are a list of options that may appear in any order in the circuit statement.

metric Allows specifications of Level 1 and Level 2 metrics for each circuit. Only the default metric type is supported. IS-IS metrics must be in the range 1 to 63. If no metric is set for the circuit, the default value is 63.

priority Determines designated router election results; higher values give a higher likelihood of becoming the designated router. The level defaults to Level 1. If no priority is specified, priority is set to a random value between 0 and 127.

On a level 2 IS, to configure a circuit with a Level 1 metric of 10 and a Level 2 metric of 20, add two metric options to the circuit statement.

The default Level is 1: the default metric is 63. The default preference for IS-IS Level 1 is 15 for IS-IS Level 2 is 18.

Tracing options

Traceoptions can be one or more of the following:

```
all
iih
lanadj
p2padj
lspdb
lspcontent
lspinput
flooding
buildlsp
csnp
psnp
route
```

```
update
paths
spf
events
```

The OSPF Statement

```
ospf yes | no | on | off [{
    defaults {
        preference preference ;
        cost cost ;
        tag [as ] tag ;
        type 1 | 2 ;
    } ;
    exportlimit routes ;
    exportinterval time ;
    traceoptions trace_options ;
    monitorauthkey authkey ;
    monitorauth none | ( [simple | md5 ] authkey ) ;
    backbone | ( area area ) {
        authtype 0 | 1 | none | simple ;
        stub [cost cost] ;
        networks {
            network [restrict ] ;
            network mask mask [restrict ] ;
            network masklen number [restrict ] ;
            host host [restrict ] ;
        } ;
        stubhosts {
            host cost cost ;
        } ;
        interface interface_list; [cost cost ] {
            interface_parameters
        } ;
        interface interface_list nonbroadcast [cost cost ] {
            pollinterval time ;
            routers {
                gateway [eligible ] ;
            } ;
            interface_parameters
        } ;
        Backbone only:
        virtuallink neighborid router_id transitarea area {
            interface_parameters
        } ;
    };
} ];
```

The following are the *interface_parameters* referred to above. They may be specified on any class of interface and are described under the `interface` clause.

```
    enable | disable;
    retransmitinterval time;
    transitdelay time;
    priority priority;
    hellointerval time;
    routerdeadinterval time;
    authkey auth_key;
```

defaults

These parameters specify the defaults used when importing OSPF ASE routes into the gated routing table and exporting routes from the gated routing table into OSPF ASEs.

preference *preference*
: **Preference** is used to determine how OSPF routes compete with routes from other protocols in the gated routing table. The default value is 150.

cost *cost*
: **Cost** is used when exporting a non-OSPF route from the gated routing table into OSPF as an ASE. The default value is 1. This may be explicitly overridden in export policy.

tag [**as**] *tag*
: OSPF ASE routes have a 32 bit tag field that is not used by the OSPF protocol, but may be used by export policy to filter routes. When OSPF is interacting with an EGP, the tag field may be used to propagate AS path information, in which case the `as` keyword is specified and the tag is limited to 12 bits of information. If not specified, the tag is set to zero.

type *1* | *2*
: Routes exported from the gated routing table into OSPF default to becoming type 1 ASEs. This default may be explicitly changed here and overridden in export policy.

ASE export rate

Because of the nature of OSPF, the rate at which ASEs are flooded must be limited. These two parameters can be used to adjust those rate limits.

exportinterval *time*
: This specifies how often a batch of ASE link state advertisements will be generated and flooded into OSPF. The default is once per second.

exportlimit *routes*
: This parameter specifies how many ASEs will be generated and flooded in each batch. The default is 100.

traceoptions *trace_options*

Specifies the tracing options for OSPF. (See Trace Statements and the OSPF specific tracing options below.)

monitorauthkey *authkey*

OSPF state may be queried using the **ospf_monitor** command utility. This utility sends non-standard OSPF packets that generate a text response from OSPF. By default, these requests are not authenticated if an authentication key is configured, the incoming requests must match the specified authentication key. No OSPF state may be changed by these packets, but the act of querying OSPF can utilize system resources.

backbonearea *area*

Each OSPF router must be configured into at least one OSPF area. If more than one area is configured, at least one must be `backbone`. The `backbone` may only be configured using the **backbone** keyword, it may not be specified as `area 0`. The backbone interface may be a `virtuallink`.

authtype *0 | 1 | none | simple*
OSPF specifies an authentication scheme per area. Each interface in the area must use this same authentication scheme although it may use a different `authenticationkey`. The currently valid values are `none` (`0`) for no authentication, or `simple` (`1`) for simple password authentication.

stub [**cost** *cost*]
A `stub` area is one in which there are no ASE routes. If a `cost` is specified, this is used to inject a default route into the area with the specified cost.

networks
The `networks` list describes the scope of an area. Intra-area LSAs that fall within the specified ranges are not advertised into other areas as inter-area routes. Instead, the specified ranges are advertised as `summary network` LSAs. If **restrict** is specified, the summary network LSAs are not advertised. Intra-area LSAs that do not fall into any range are also advertised as summary network LSAs. This option is very useful on well designed networks in reducing the amount of routing information propagated between areas. The entries in this list are either networks, or a subnetwork/mask pair. See the section on Route Filtering for more detail about specifying ranges.

stubhosts
This list specifies directly attached hosts that should be advertised as reachable from this router and the costs they should be advertised with. Point-to-point interfaces on which it is not desirable to run OSPF should be specified here.

It is also useful to assign an additional address to the loopback interface (one not on the 127 network) and advertise it as a stub hosts. If this address is the same one used as the router-id, it enables routing to OSPF routers by router-id, instead of by an interface address. This is more reliable than routing to one of the router’s interface addresses that may not always be reachable.

interface *interface_list* [**cost** *cost*]

This form of the interface clause is used to configure a `broadcast` (which requires IP multicast support) or a `point-to-point` interface. See the section on interface list specification for the description of the *interface_list*.

Each interface has a `cost`. The costs of all interfaces a packet must cross to reach a destination are summed to get the cost to that destination. The default cost is one, but another non-zero value may be specified.

Interface parameters common to all types of interfaces are:

retransmitinterval *time*
: The number of seconds between link state advertisement retransmissions for adjacencies belonging to this interface.

transitdelay *time*
: The estimated number of seconds required to transmit a link state update over this interface. **Transitdelay** takes into account transmission and propagation delays and must be greater than 0.

priority *priority*
: A number between 0 and 255 specifying the priority for becoming the designated router on this interface. When two routers attached to a network both attempt to become the designated router, the one with the highest priority wins. A router whose router priority is set to 0 is ineligible to become the designated router.

hellointerval *time*
: The length of time, in seconds, between Hello packets that the router sends on the interface.

routerdeadinterval *time*
: The number of seconds not hearing a router's Hello packets before the router's neighbors will declare it down.

authkey *auth_key*
: Used by OSPF authentication to generate and verify the authentication field in the OSPF header. The authentication key can be configured on a per interface basis. It is specified by one to eight decimal digits separated by periods, a one to eight byte hexadecimal string preceded by `0x`, or a one to eight character string in double quotes.

Point-to-point interfaces also support this additional parameter:

nomulticast
: By default, OSPF packets to neighbors on point-to-point interfaces are sent via the IP multicast mechanism. If the use of IP multicasting is not desired, the *nomulticast* parameter may be specified to force the use of unicast OSPF packets. **gated.conf** will detect this condition and fall back to using sending unicast OSPF packets to this point-to-point neighbor.

If the use of IP multicasting is not desired because the remote neighbor does not support it, the *nomulticast* parameter may be specified to force the use of unicast OSPF packets. This option may also be used to eliminate warnings when **gated.conf** detects the bug mentioned above.

interface *interface_list* **nonbroadcast** [**cost** *cost*]	This form of the interface clause is used to specify a `nonbroadcast` interface on a `non-broadcast multi-access` (NBMA) media. Since an OSPF `broadcast` media must support IP multicasting, a broadcast-capable media, such as Ethernet, that does not support IP multicasting must be configured as a non-broadcast interface. A non-broadcast interface supports any of the standard `interface` clauses listed above, plus the following two that are specific to non-broadcast interfaces: **pollinterval** *time* Before adjacency is established with a neighbor, OSPF packets are sent periodically at the specified `pollinterval`. **routers** By definition, it is not possible to send broadcast packets to discover OSPF neighbors on a non-broadcast, so all neighbors must be configured. The list includes one or more neighbors and an indication of their eligibility to become a designated router.
virtuallink neighborid *router_id* **transitarea** *area*	Virtual links are used to establish or increase connectivity of the backbone area. The `neighborid` is the `router_id` of the other end of the virtual link. The transit `area` specified must also be configured on this system. All standard interface parameters defined by the `interface` clause above may be specified on a virtual link.

Tracing options

In addition to the following OSPF specific trace flags, OSPF supports the `state` that traces interface and neighbor state machine transitions.

lsabuild Link State Advertisement creation

spf Shortest Path First (SPF) calculations

Packet tracing options (which may be modified with `detail`, `send` and `recv`):

hello OSPF `HELLO` packets that are used to determine neighbor reachability.

dd OSPF Database Description packets that are used in synchronizing OSPF databases.

request OSPF Link State Request packets that are used in synchronizing OSPF databases.

lsu OSPF Link State Update packets that are used in synchronizing OSPF databases.

ack OSPF Link State Ack packets that are used in synchronizing OSPF databases.

The EGP Statement

```
EGP yes | no | on | off
[{
    preference preference ;
    defaultmetric metric ;
    packetsize number ;
```

```
        traceoptions trace_options ;
        group
            [peeras autonomous_system ]
            [localas autonomous_system ]
            [maxup number ]
        {
            neighbor host
                [metricout metric ]
                [preference preference ]
                [preference2 preference ]
                [ttl ttl ]
                [nogendefault ]
                [importdefault ]
                [exportdefault ]
                [gateway gateway ]
                [lcladdr local_address ]
                [sourcenet network ]
                [minhello | p1 time ]
                [minpoll | p2 time ]
                [traceoptions trace_options ]
                ;
       } ;
    } ] ;
```

preference *preference*	Sets the preference for routes learned from RIP. The default preference is 200. This preference may be overridden by a preference specified on the `group` or `neighbor` statements or by import policy.
defaultmetric *metric* **;**	Defines the metric used when advertising routes via EGP. If not specified, the default value is 255 that some systems may consider unreachable. This choice of values requires you to explicitly specify a metric when exporting routes to EGP neighbors. This metric may be overridden by a metric specified on the neighbor or group statements or in export policy.
packetsize *maxpacketsize*	This defines the expected maximum size of a packet that EGP expects to receive from this neighbor. If a packet larger than this value is received, it will be incomplete and have to be discarded. The length of this packet will be noted and the expected size will be increased to be able to receive a packet of this size. Specifying the parameter here will prevent the first packet from being dropped. If not specified, the default size is 8192 bytes. All packet sizes are rounded up to a multiple of the system page size.
traceoptions *trace_options*	Specifies the tracing options for EGP. By default these are inherited from the global trace options. These values may be overridden on a group or neighbor basis. (See Trace Statements and the EGP specific tracing options below.)

group

EGP neighbors must be specified as members of a `group`. A group is usually used to group all neighbors in one autonomous system. Parameters specified on the group clause apply to all of the subsidiary neighbors unless explicitly overridden on a neighbor clause. Any number of `group` clauses may specify any number of `neighbor` clauses.

Any parameters from the `neighbor` subclause may be specified on the `group` clause to provide defaults for the whole group (which may be overridden for individual neighbors). In addition, the `group` clause is the only place to set the following attributes:

peeras
: Identifies the autonomous system number expected from peers in the group. If not specified, it will be learned dynamically.

localas
: Identifies the autonomous system that **gated.conf** is representing to the group. The default is that which has been set globally in the `autonomoussystem` statement. This option is usually only used when *masquerading* as another autonomous system and its use is discouraged.

maxup
: Specifies the number of neighbors the **gated** daemon should acquire from this group. The default is to acquire all of the neighbors in the group. The **gated** daemon will attempt to acquire the first `maxup` neighbors in the order listed. If one of the first neighbors is not available, it will acquire one further down the list. If after start-up the **gated** daemon does manage to acquire the more desirable neighbor, it will drop the less desirable one.

neighbor *neighbor_address*

Each neighbor subclause defines one EGP neighbor within a group. The only part of the subclause that is required is the `neighbor_address` argument that is the symbolic host name or IP address of the neighbor. All other parameters are optional.

preference *preference*
: Specifies the preference used for routes learned from these neighbors. This can differ from the default EGP preference set in the `EGP` statement, so that the **gated** daemon can prefer routes from one neighbor, or group of neighbors, over another. This preference may be explicitly overridden by import policy.

preference2 *preference*
: In the case of a `preference` tie, the second preference, `preference2` may be used to break the tie. The default value is 0.

metricout *metric*
: This defines a metric to be used for all routes sent to this neighbor. The value overrides the default metric set in the `EGP` statement and any metrics specified by export policy, but only for this specific neighbor or group of neighbors.

nogendefault
: Prevents **gated.conf** from generating a default route when EGP receives a valid update from its neighbor. The default route is only generated when the **gendefault** option is enabled.

importdefault
: Enables the **gated** daemon to accept the default route (0.0.0.0) if it is included in a received EGP update. If not specified, the default route contained in an EGP update is ignored. For efficiency, some networks have external routers announce a default route to avoid sending large EGP update packets.

exportdefault
: Enables the **gated** daemon to include the default route (0.0.0.0) in EGP updates sent to this EGP neighbor. This allows the system to advertise the default route via EGP. Normally a default route is not included in EGP updates.

gateway *gateway*
: If a network is not shared with a neighbor, `gateway` specifies a router on an attached network to be used as the next hop router for routes received from this neighbor. This option is only rarely used.

lcladdr *local_address*
: Specifies the address to be used on the local end of the connection with the neighbor. The local address must be on an interface that is shared with the neighbor or with the neighbor's *gateway* when the `gateway` parameter is used. A session will only be opened when an interface with the appropriate local address (through which the neighbor or gateway address is directly reachable) is operating.

sourcenet *network*
: Specifies the network queried in the EGP Poll packets. By default, this is the network shared with the neighbor's address specified. If there is no network shared with the neighbor, one of the networks the neighbor is attached to should be specified. This parameter can also be used to specify a network shared with the neighbor other than the one on which the EGP packets are sent. This parameter is normally not needed.

p1 *time*

minhello *time*
: Sets the minimum acceptable interval between the transmission of EGP `HELLO` packets. The default hello interval is 30 seconds. If the neighbor fails to respond to three hello packets, the **gated** daemon stops trying to acquire the neighbor. Setting a larger interval gives the neighbor a better chance to respond. **Minhello** is an alias for the `P1` value defined in the EGP specification.

p2 *time*

minpoll *time*
: Sets the time interval between polls to the neighbor. The default is 120 seconds. If three polls are sent without a response, the neighbor is declared "down" and all routes learned from that neighbor are removed from the routing database. A longer polling interval supports a more stable routing database but is not as responsive to routing changes. **Minpoll** is an alias for the `P2` value defined in the EGP specification.

ttl *ttl*
: By default, the **gated** daemon sets the IP TTL for local neighbors to *one* and the TTL for non-local neighbors to 255. This option is provided when attempting to communicate with improperly functioning routers that ignore packets sent with a TTL of one.

traceoptions *trace_options*
: Specifies the tracing options for this EGP neighbor. By default, these are inherited from group or EGP global trace options. (See Trace Statements and the EGP specific tracing options below.)

Tracing options

The `state` and `policy` options work with EGP.

Packet tracing options (which may be modified with `detail`, `send` and `recv`):

packets All EGP packets

hello EGP `HELLO/I-HEARD-U` packets that are used to determine neighbor reachability.

acquire EGP `ACQUIRE/CEASE` packets that are used to initiate and terminate EGP sessions.

update EGP `POLL/UPDATE` packets that are used to request and receive reachability updates.

The BGP Statement

```
 bgp   yes  |  no  |  on  |  off
[ {
    preference   preference   ;
    defaultmetric   metric   ;
    traceoptions   trace_options   ;
    group   type  (  external   peeras   autonomous_system  )
       | (  internal   peeras   autonomous_system  )
       | (  IGP   peeras   autonomous_system   proto   proto  )
       | (  routing   peeras   autonomous_system   proto   proto
                interface   interface_list  )
       | (  test   peeras   autonomous_system  )
    {
        allow   {
            network
            network   mask   mask
            network   masklen   number
            all
            host   host
        }   ;
        peer   host
           [ metricout   metric  ]
           [ localas   autonomous_system  ]
           [ nogendefault]
           [ gateway   gateway  ]
           [ preference   preference  ]
           [ preference2  preference  ]
           [ lcladdr   local_address  ]
           [ holdtime   time  ]
           [ version   number  ]
           [ passive]
           [ indelay   time  ]
           [ outdelay   time  ]
           [ keep  [ all  |  none] ]
           [ noaggregatorid]
           [ keepalivesalways]
           [ v3asloopokay]
           [ nov4asloop]
           [ logupdown]
           [ ttl   ttl  ]
           [ traceoptions   trace_options  ]
            ;
```

```
        }   ;
     }]  ;
```

```
external  |  internal  |  IGP  |  test
```

The `bgp` statement enables or disables BGP. By default, BGP is disabled. The default metric for announcing routes via BGP is not to send a metric.

preference *preference*	Sets the preference for routes learned from RIP. The default preference is 170. This preference may be overridden by a preference specified on the `group` or `peer` statements or by import policy.
defaultmetric *metric*	Defines the metric used when advertising routes via BGP. If not specified, no metric is propagated. This metric may be overridden by a metric specified on the neighbor or group statements or in export policy.
traceoptions *trace_options*	Specifies the tracing options for BGP. By default these are inherited from the global trace options. These values may be overridden on a group or neighbor basis. (See Trace Statements and the BGP specific tracing options below.)

Groups

BGP peers are grouped by type and the autonomous system of the peers. Any number of groups may be specified, but each must have a unique combination of type and peer autonomous system. There are four possible group types:

group type external peeras *autonomous_system*
In the classic external BGP group, full policy checking is applied to all incoming and outgoing advertisements. The external neighbors must be directly reachable through one of the machine's local interfaces. By default no metric is included in external advertisements, and the next hop is computed with respect to the shared interface.

group type internal peeras *autonomous_system*
An internal group operating where there is no IP-level IGP. All neighbors in this group are required to be directly reachable via a single interface. All next hop information is computed with respect to this interface. Import and export policy may be applied to group advertisements. Routes received from external BGP or EGP neighbors are by default readvertised with the received metric.

group type IGP peeras *autonomous_system* **proto** *proto*
An internal group that runs in association with an interior protocol. The IGP group examines routes that the IGP is exporting and sends an advertisement only if the path attributes could not be entirely represented in the IGP tag mechanism. Only the AS path, path origin, and transitive optional attributes are sent with routes. No metric is sent, and the next hop is set to the local address used by the connection. Received internal BGP routes are not used or readvertised. Instead, the AS path information is attached to the corresponding IGP route and the latter is used for readvertisement. Since internal IGP peers are sent only a subset of the routes that the IGP is exporting, the export policy used is the IGP's. There is no need to implement the "don't routes from peers in the same group" constraint since the advertised routes are routes that IGP already exports.

group type routing peeras *autonomous_system* **proto** *proto* **interface** *interface_list*

An internal group that uses the routes of an interior protocol to resolve forwarding addresses. A type routing group propagates external routes between routers that are not directly connected, and computes immediate next hops for these routes by using the BGP next hop that arrived with the route as a forwarding address to be resolved via an internal protocol's routing information. In essence, internal BGP is used to carry AS external routes, while the IGP is expected to only carry AS internal routes, and the latter is used to find immediate next hops for the former.

The *proto* names the interior protocol to be used to resolve BGP route next hops, and may be the name of any IGP in the configuration. By default, the next hop in BGP routes advertised to type routing peers will be set to the local address on the BGP connection to those peers, as it is assumed a route to this address will be propagated via the IGP. The *interface_list* can optionally provide list interfaces whose routes are carried via the IGP for which third party next hops may be used instead.

group type test peeras *autonomous_system*

An extension to external BGP that implements a fixed policy using test peers. Fixed policy and `special case` code make test peers relatively inexpensive to maintain. Test peers do not need to be on a directly attached network. If the **gated** daemon and the peer are on the same (directly attached) subnet, the advertised next hop is computed with respect to that network, otherwise the next hop is the local machine's current next hop. All routing information advertised by and received from a test peer is discarded, and all BGP advertisable routes are sent back to the test peer. Metrics from EGP- and BGP-derived routes are forwarded in the advertisement, otherwise no metric is included.

Group parameters

The BGP statement has `group` clauses and `peer` subclauses. Any number of peer subclauses may be specified within a group. A group clause usually defines default parameters for a group of peers, these parameters apply to all subsidiary peer subclauses. Any parameters from the peer subclause may be specified on the group clause to provide defaults for the whole group (which may be overridden for individual peers).

Specifying peers

Within a group, BGP peers may be configured in one of two ways. They may be explicitly configured with a `peer` statement, or implicitly configured with the `allow` statement. Both are described here:

allow
: The allow clause allows for `peer` connections from any addresses in the specified range of network and mask pairs. All parameters for these peers must be configured on the group clause. The internal peer structures are created when an incoming open request is received and destroyed when the connection is broken. For more detail on specifying the network/mask pairs, see the section on Route Filtering.

peer *host*
: A `peer` clause configures an individual peer. Each peer inherits all parameters specified on a group as defaults. Those defaults may be overridden by parameters explicitly specified on the peer subclause.

Within each `group` clause, individual peers can be specified or a group of *potential* peers can be specified using `allow`. `Allow` is used to specify a set of address masks. If the **gated** daemon receives a BGP connection request from any address in the set specified, it will accept it and set up a peer relationship.

Peer parameters

The BGP `peer` subclause allows the following parameters, which can also be specified on the `group` clause. All are optional.

metricout *metric*
: If specified, this metric is used as the primary metric on all routes sent to the specified peer(s). This metric overrides the default metric, a metric specified on the group and any metric specified by export policy.

localas *autonomous_system*
: Identifies the autonomous system that the **gated** daemon is representing to this group of peers. The default is that which has been set globally in the `autonomoussystem` statement.

nogendefault
: Prevents **gated.conf** from generating a default route when EGP receives a valid update from its neighbor. The default route is only generated when the **gendefault** option is enabled.

gateway *gateway*
: If a network is not shared with a peer, `gateway` specifies a router on an attached network to be used as the next hop router for routes received from this neighbor. This parameter is not needed in most cases.

preference *preference*
: Specifies the preference used for routes learned from these peers. This can differ from the default BGP preference set in the `bgp` statement, so that the **gated** daemon can prefer routes from one peer, or group of peer, over others. This preference may be explicitly overridden by import policy.

preference2 *preference*
: In the case of a `preference` tie, the second preference, `preference2` may be used to break the tie. The default value is 0.

lcladdr *local_address*
: Specifies the address to be used on the local end of the TCP connection with the peer. For `external` peers the local address must be on an interface that is shared with the peer or with the peer's `gateway` when the `gateway` parameter is used. A session with an external peer will only be opened when an interface with the appropriate local address (through which the peer or gateway address is directly reachable) is operating. For other types of peers, a peer session will be maintained when any interface with the specified local address is operating. In either case, incoming connections will only be recognized as matching a configured peer if they are addressed to the configured local address.

holdtime *time*
: Specifies the BGP holdtime value to use when negotiating the connection with this peer, in seconds. According to BGP, if the **gated** daemon does not receive a keepalive, update, or notification message within the period specified in the Hold Time field of the BGP Open message, then the BGP connection will be closed. The value must be either 0 (no keepalives will be sent) or at least 3.

version *version*
: Specifies the version of the BGP protocol to use with this peer. If not specified, the highest supported version is used first and version negotiation is attempted. If it is specified, only the

specified version will be offered during negotiation. Currently supported versions are 2, 3 and 4.

passive

Specifies that active OPENs to this peer should not be attempted. the **gated** daemon should wait for the peer to issue an OPEN. By default, all explicitly configured peers are active, they periodically send OPEN messages until the peer responds.

indelay *time*

outdelay *time*

Used to dampen route fluctuations. `Indelay` is the amount of time a route learned from a BGP peer must be stable before it is accepted into the gated routing database. `Outdelay` is the amount of time a route must be present in the gated routing database before it is exported to BGP. The default value for each is 0, meaning that these features are disabled.

keep all

Used to retain routes learned from a peer even if the routes' AS paths contain one of our exported AS numbers.

noaggregatorid

Causes the **gated** daemon to specify the routerid in the aggregator attribute as zero (instead of its routerid) in order to prevent different routers in an AS from creating aggregate routes with different AS paths.

keepalivesalways

Causes the **gated** daemon to always send keepalives, even when an update could have correctly substituted for one. This allows interoperability with routers that do not completely obey the protocol specifications on this point.

v3asloopokay

By default, the **gated** daemon will not advertise routes whose AS path is looped (that is, with an AS appearing more than once in the path) to version 3 external peers. Setting this flag removes this constraint. Ignored when set on internal groups or peers.

nov4asloop

Prevents routes with looped AS paths from being advertised to version 4 external peers. This can be useful to avoid advertising such routes to peers that would incorrectly forward the routes on to version 3 neighbors.

logupdown

Causes a message to be logged via the syslog mechanism whenever a BGP peer enters or leaves the `ESTABLISHED` state.

traceoptions *trace_options*

Specifies the tracing options for this BGP neighbor. By default, these are inherited from group or BGP global trace options.(See Trace Statements and the BGP specific tracing options below.)

Tracing options

Note: The `state` option works with BGP, but does not provide true state transition information.

Packet tracing options (which may be modified with `detail`, `send`, and `recv`):

packets	All BGP packets.
open	BGP `OPEN` packets that are used to establish a peer relationship.
update	BGP `UPDATE` packets that are used to pass network reachability information.
keepalive	BGP `KEEPALIVE` packets that are used to verify peer reachability.

The ICMP Statement

```
icmp {
    traceoptions trace_options ;
}
```

traceoptions *trace_options***;** Specifies the tracing options for ICMP. (See Trace Statements and the ICMP specific tracing options below.)

Tracing options

Packet tracing options (which may be modified with `detail` and `recv`):

packets All ICMP packets received.

redirect Only ICMP `REDIRECT` packets received.

routerdiscovery Only ICMP `ROUTER DISCOVERY` packets received.

info Only ICMP informational packets, which include mask request/response, info request/response, echo request/response, and time stamp request/response.

error Only ICMP error packets, which include time exceeded, parameter problem, unreachable and source quench.

The SNMP Statement

The Simple Network Management Protocol (SNMP) is a not a routing protocol but a network management protocol. The `snmp` statement controls whether **gated.conf** tries to contact the SNMP Multiplexing daemon to register supported variables. The SNMP daemon, `smuxd`, must be run independently. The `snmp` statement only controls whether **gated.conf** keeps the management software apprised of its status.

gated.conf communicates with the SNMP daemon via the SMUX protocol that is described in RFC 1227.

```
snmp yes | no | on | off
[ {
    port port ;
    debug;
    traceoptions traceoptions;
}] ;
```

Reporting is enabled by specifying `yes` or `on` and disabled with `no` or `off`. The default is `on`.

port *port*	By default, the SMUX daemon listens for requests on port 199. The **gated.conf** subroutine can be configured to try to contact the SMUX daemon on a different port by explicitly specifying the port.
debug	Specifying this option enables debugging of the ISODE SMUX code. The default is debugging disabled.
traceoptions *trace_options*	Specifies the tracing options for SMUX. (See Trace Statements and the SMUX specific tracing options below.)

Tracing options

There are no SNMP-specific trace options. The `detail`, `send`, and `recv` options are not supported.

receive	SNMP requests received from the SMUX daemon and the associated responses.
register	Protocol requests to register variables.
resolve	Protocol requests to resolve variable names.
trap	SNMP trap requests from protocols.

Static Statements

`Static` statements define the static routes used by the **gated** daemon. A single `static` statement can specify any number routes. The `static` statements occur after protocol statements and before control statements in the **gated.conf** file. Any number of `static` statements may be specified, each containing any number of static route definitions. These routes can be overridden by routes with better preference values.

```
static   {
   (  host   host   ) |  default   |
   (  network  [ (  mask   mask   ) | (  masklen   number   ) ] )
        gateway   gateway_list
       [ interface   interface_list  ]
       [ preference   preference  ]
       [ retain]
       [ reject]
       [ blackhole]
       [ noinstall] ;
   (  network  [ (  mask   mask   ) | (  masklen   number   ) ] )
        interface   interface
       [ preference  preference  ]
       [ retain]
       [ reject]
       [ blackhole]
       [ noinstall] ;
}   ;
```

host *host* **gateway** *gateway_list* (*network* [(**mask** *mask*) | (**masklen** *number*)]) **default gateway** *gateway_list*

This is the most general form of the static statement. It defines a static route through one or more gateways. Static routes are installed when one or more of the `gateways` listed are available on directly attached interfaces.

Parameters for static routes are:

interface *interface_list*	When this parameter is specified, gateways are only considered valid when they are on one of these interfaces. See the section on interface_list specification for the description of the *interface_list*.
preference *preference*	This option selects the preference of this static route. The preference controls how this route competes with routes from other protocols. The default preference is 60.
retain	Normally the **gated** daemon removes all routes except interface routes from the kernel forwarding table during a graceful shutdown. The `retain` option may be used to prevent specific static routes from being removed. This is useful to insure that some routing is available when **gated** is not running.
reject	Instead of forwarding a packet like a normal route, `reject` routes cause packets to be dropped and `unreachable` messages to be sent to the packet originators. Specifying this option causes this route to be installed as a reject route. Not all kernel forwarding engines support reject routes.
blackhole	A `blackhole` route is the same as a `reject` route except that `unreachable` messages are not supported.
noinstall	Normally the route with the lowest preference is installed in the kernel forwarding table and is the route exported to other protocols. When `noinstall` is specified on a route, it will not be installed in the kernel forwarding table when it is active, but it will still be eligible to be exported to other protocols.

(*network* [(**mask** *mask*) | (**masklen** *number*)]) **interface** *interface*

This form defines a static interface route that is used for primitive support of multiple network addresses on one interface. The `preference`, `retain`, `reject`, `blackhole` and `noinstall` options are the same as described above.

The Import Statement

Importation of routes from routing protocols and installation of the routes in the **gated** daemon's routing database is controlled by `import` statements. The format of an `import` statement varies depending on the source protocol.

Specifying preferences

In all cases, one of two keywords may be specified to control how routes compete with other protocols:

```
restrict
preference preference
```

restrict Specifies that the routes are not desired in the routing table. In some cases, this means that the routes are not installed in the routing table. In others, it means that they are installed with a negative preference; this prevents them from becoming *active* so they will not be installed in the forwarding table, or exported to other protocols.

preference *preference* Specifies the preference value used when comparing this route to other routes from other protocols. The route with the lowest preference available at any given route becomes the *active* route, is installed in the forwarding table, and is eligible to be exported to other protocols. The default preferences are configured by the individual protocols.

Route Filters

All the formats allow route filters as shown below. See the section on route filters for a detailed explanation of how they work. When no route filtering is specified (that is, when `restrict` is specified on the first line of a statement), all routes from the specified source will match that statement. If any filters are specified, only routes that match the specified filters will be imported. Put differently, if any filters are specified, an `all restrict;` is assumed at the end of the list.

```
network [ exact | refines ]
network mask mask [exact | refines ]
network masklen number [ exact | refines ]
default
host host
```

Importing Routes from BGP and EGP

```
import proto bgp | EGP autonomoussystem autonomous_system
    restrict ;
import proto bgp | EGP autonomoussystem autonomous_system
    [ preference preference ] {
    route_filter [ restrict | ( preference preference ) ] ;
} ;

import proto bgp aspath aspath_regexp
    origin any | ( [ IGP ] [EGP ] [ incomplete ] )
    restrict ;
import proto bgp aspath aspath_regexp
    origin any | ( [ IGP ] [EGP ] [ incomplete ] )
    [ preference preference ] {
    route_filter [ restrict | ( preference preference ) ] ;
} ;
```

EGP importation may be controlled by autonomous system.

BGP also supports controlling propagation by the use of AS path regular expressions, which are documented in the section on Matching AS paths.

Note: EGP and BGP versions 2 and 3 only support the propagation of *natural* networks, so the `host` and `default` route filters are meaningless. BGP version 4 supports the propagation of any destination along with a *contiguous* network mask.

EGP and BGP both store any routes that were rejected implicitly by not being mentioned in a route filter, or explicitly with the `restrict` keyword in the routing table with a negative preference. A negative preference prevents a route from becoming active, which prevents it from being installed in the forwarding table, or exported to other protocols. This alleviates the need to break and re-establish a session upon reconfiguration if importation policy is changed.

Importing Routes from RIP, HELLO and Redirects

```
import proto rip | hello | redirect
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    restrict ;
import proto rip | hello | redirect
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    [ preference preference ] {
    route_filter [ restrict | ( preference preference ) ] ;
} ;
```

The importation of RIP, HELLO, and Redirect routes may be controlled by any of protocol, source interface, and source gateway. If more than one is specified, they are processed from most general (protocol) to most specific (gateway).

RIP and HELLO don’t support the use of preference to choose between routes of the same protocol. That is left to the protocol metrics. These protocols do not save routes that were rejected since they have short update intervals.

Importing Routes from OSPF

```
import proto ospfase [ tag ospf_tag ] restrict ;
import proto ospfase [ tag ospf_tag ]
    [ preference preference ] {
     route_filter [ restrict | ( preference preference ) ] ;
} ;
```

Due to the nature of OSPF, only the importation of ASE routes may be controlled. OSPF intra- and inter-area routes are always imported into the gated routing table with a preference of 10. If a tag is specified, the import clause will only apply to routes with the specified tag.

It is only possible to restrict the importation of OSPF ASE routes when functioning as an AS border router. This is accomplished by specifying an **export ospfase** clause. Specification of an empty export clause may be used to restrict importation of ASEs when no ASEs are being exported.

Like the other interior protocols, preference can not be used to choose between OSPF ASE routes, that is done by the OSPF costs. Routes that are rejected by policy are stored in the table with a negative preference.

The Export Statement

The `import` statement controls routes received from other systems that are used by the **gated** daemon, and the `export` statement controls which routes are advertised by the **gated** daemon to other systems. Like the `import` statement, the syntax of the `export` statement varies slightly per protocol. The syntax of the `export` statement is similar to the syntax of the `import` statement, and the meanings of many of the parameters are identical. The main difference between the two is that while route importation is just controlled by source information, route exportation is controlled by both destination and source.

The outer portion of a given `export` statement specifies the destination of the routing information you are controlling. The middle portion restricts the sources of importation that you wish to consider And the innermost portion is a route filter used to select individual routes.

Specifying Metrics

The most specific specification of a metric is the one applied to the route being exported. The values that may be specified for a metric depend on the destination protocol that is referenced by this export statement.

```
restrict
metric metric
```

restrict Specifies that nothing should be exported. If specified on the destination portion of the `export` statement, it specifies that nothing at all should be exported to this destination. If specified on the source portion, it specifies that nothing from this source should be exported to this destination. If specified as part of a route filter, it specifies that the routes matching that filter should not be exported.

metric *metric* Specifies the metric to be used when exporting to the specified destination.

Route Filters

All the formats allow route filters as shown below. See the section on route filters for a detailed explanation of how they work. When no route filtering is specified (that is, when `restrict` is specified on the first line of a statement), all routes from the specified source will match that statement. If any filters are specified, only routes that match the specified filters will be exported. Put differently, if any filters are specified, an `all restrict ;` is assumed at the end of the list.

```
network [ exact | refines ]
network mask mask [exact | refines ]
network masklen number [ exact | refines ]
default
host host
```

Specifying the Destination

As mentioned above, the syntax of the `export` statement varies depending on the protocol to which it is being applied. One thing that applies in all cases is the specification of a metric. All protocols define a default metric to be used for routes being exported, in most cases this can be overridden at several levels of the export statement.

The specification of the source of the routing information being exported (the export_list) is described below.

Exporting to EGP and BGP

```
export proto bgp | EGP as autonomous system
    restrict ;
export proto bgp | EGP as autonomous system
    [ metric metric ] {
    export_list ;
} ;
```

Exportation to EGP and BGP is controlled by autonomous system, the same policy is applied to all routers in the AS.

EGP metrics range from 0 to 255 inclusive with 0 being the most attractive.

BGP metrics are 16 bit unsigned quantities, that is, they range from 0 to 65535 inclusive with 0 being the most attractive.

If no export policy is specified, only routes to attached interfaces will be exported. If any policy is specified, the defaults are overridden. It is necessary to explicitly specify everything that should be exported.

> **Note:** EGP and BGP versions 2 and 3 only support the propagation of *natural* networks, so the `host` and `default` route filters are meaningless. BGP version 4 supports the propagation of any destination along with a *contiguous* network mask.

Exporting to RIP and HELLO

```
export proto rip | hello
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    restrict ;
export proto rip | hello
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    [ metric metric ] {
    export_list ;
} ;
```

Exportation to RIP and HELLO is controlled by any of protocol, interface or gateway. If more than one is specified, they are processed from the most general (protocol) to the most specific (gateway).

It is not possible to set metrics for exporting RIP routes into RIP, or exporting HELLO routes into HELLO. Attempts to do this are silently ignored.

If no export policy is specified, RIP and interface routes are exported into RIP and HELLO and interface routes are exported into HELLO. If any policy is specified, the defaults are overridden. It is necessary to explicitly specify everything that should be exports.

RIP version 1 and HELLO assume that all subnets of the shared network have the same subnet mask so they are only able to propagate subnets of that network. RIP version 2 removes that restriction and is capable of propagating all routes when not sending version 1 compatible updates.

To announce routes that specify a next hop of the loopback interface (that is, static and internally generated default routes) via RIP or HELLO, it is necessary to specify the metric at some level in the `export` clause. For example, just setting a default metric for RIP or HELLO is not sufficient. This is a safeguard to verify that the announcement is intended.

Exporting to OSPF

```
export proto osfpase [ type 1 | 2 ] [ tag ospf_tag ]
    restrict ;
export proto osfpase [ type 1 | 2 ] [ tag ospf_tag ]
    [ metric metric ] {
    export_list ;
} ;
```

It is not possible to create OSPF intra- or inter-area routes by exporting routes from the the **gated** daemon routing table into OSPF. It is only possible to export from the **gated** daemon routing table into OSPF ASE routes. It is also not possible to control the propagation of OSPF routes within the OSPF protocol.

There are two types of OSPF ASE routes, *type 1* and *type 2*. See the OSPF protocol configuration for a detailed explanation of the two types. The default type is specified by the `defaults` subclause of the `ospf` clause. This may be overridden by a specification on the `export` statement.

OSPF ASE routes also have the provision to carry a *tag*. This is an arbitrary 32 bit number that can be used on OSPF routers to filter routing information. See the OSPF protocol configuration for detailed information on OSPF tags. The default tag specified by the `ospf defaults` clause may be overridden by a tag specified on the `export` statement.

Specifying the Source

The export list specifies **export** based on the origin of a route and the syntax varies depending on the source.

Exporting BGP and EGP Routes

```
proto bgp | EGP autonomoussystem autonomous_system
    restrict ;
proto bgp | EGP autonomoussystem autonomous_system
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

BGP and EGP routes may be specified by the source autonomous system. All routes may be exported by as path, see the Exporting by AS Path section for more information.

Exporting RIP and HELLO Routes

```
proto rip | hello
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    restrict ;
proto rip | hello
```

```
    [ ( interface interface_list ) | (gateway gateway_list ) ]
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

RIP and HELLO routes may be exported by protocol, source interface, and/or source gateway.

Exporting OSPF Routes

```
proto ospf | ospfase restrict ;
proto ospf | ospfase [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

Both OSPF and OSPF ASE routes may be exported into other protocols. See below for information on exporting by tag.

Exporting Routes from Non-routing Protocols

Non-routing with Interface

```
proto direct | static | kernel
    [ (interface interface_list ) ]
    restrict ;
proto direct | static | kernel
    [ (interface interface_list ) ]
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

These protocols may be exported by protocol, or by the interface of the next hop. These protocols are:

direct Routes to directly attached interfaces.

static Static routes specified in a `static` clause.

kernel Routes learned from the routing socket are installed in the gated routing table with a protocol of *kernel*. These routes may be exported by referencing this protocol.

Non-routing by Protocol

```
proto default | aggregate
    restrict ;
proto default | aggregate
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

These protocols may only be referenced by protocol.

default Refers to routes created by the `gendefault` option. It is recommended that route generation be used instead.

aggregate Refers to routes synthesized from other routes when the `aggregate` and `generate` statements are used. See the section on Route Aggregation for more information.

Exporting by AS Path

```
proto proto | all aspath aspath_regexp
    origin any | ( [ IGP ] [EGP ] [ incomplete ] )
    restrict ;
proto proto | all aspath aspath_regexp
    origin any | ( [ IGP ] [EGP ] [ incomplete ] )
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

When BGP is configured, all routes are assigned an AS path when they are added to the routing table. For all interior routes, this AS path specifies IGP as the origin and no ASEs in the AS path (the current AS is added when the route is exported). For EGP routes this AS path specifies EGP as the origin and the source AS as the AS path. For BGP routes, the AS path is stored as learned from BGP.

AS path regular expressions are documented in the section on Matching AS paths.

Exporting by Route Tag

```
proto proto | all tag tag restrict ;
proto proto | all tag tag
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

Both OSPF and RIP version 2 currently support tags; all other protocols always have a tag of zero. The source of exported routes may be selected based on this tag. This is useful when routes are classified by a tag when they are exported into a given routing protocol.

Route Aggregation

Route aggregation is a method of generating a more general route given the presence of a specific route. It is used, for example, at an autonomous system border to generate a route to a network to be advertised via EGP given the presence of one or more subnets of that network learned via RIP. No aggregation is performed unless explicitly requested in an `aggregate` statement.

Route aggregation is also used by regional and national networks to reduce the amount of routing information passed around. With careful allocation of network addresses to clients, regional networks can just announce one route to regional networks instead of hundreds.

Aggregate routes are not actually used for packet forwarding by the originator of the aggregate route, only by the receiver (if it wishes).

A slight variation of aggregation is the generation of a route based on the existence of certain conditions. This is sometimes known as the *route of last resort*. This route inherits the next hops and aspath from the contributor specified with the lowest (most favorable) preference. The most common usage for this is to generate a default based on the presence of a route from a peer on a neighboring backbone.

Aggregation and Generation syntax

```
aggregate default
    | ( network [ ( mask mask ) | ( masklen number ) ] )
    [ preference preference ] [ brief ] {
    proto [ all | direct | static | kernel | aggregate | proto ]
        [ ( as autonomous system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        restrict ;
    proto [ all | direct | static | kernel | aggregate | proto ]
        [ ( as autonomous system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        [ preference preference ] {
        route_filter [ restrict | ( preference preference ) ] ;
    } ;
} ;

generate default
    | ( network [ ( mask mask ) | ( masklen number ) ] )
    [ preference preference ] {
        [ ( as autonomous system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        restrict ;
    proto [ all | direct | static | kernel | aggregate | proto ]
        [ ( as autonomous system ) | ( tag tag )
            | ( aspath aspath_regexp ) ]
        [ preference preference ] {
        route_filter [ restrict | ( preference preference ) ] ;
    } ;
} ;
```

Routes that match the route filters are called *contributing* routes. They are ordered according to the aggregation preference that applies to them. If there are more than one contributing routes with the same aggregating preference, the route's own preferences are used to order the routes. The preference of the aggregate route will be that of contributing route with the lowest aggregate preference.

preference *preference*	Specifies the preference to assign to the resulting aggregate route. The default preference is 130.
brief	Used to specify that the AS path should be truncated to the longest common AS path. The default is to build an AS path consisting of SETs and SEQUENCEs of all contributing AS paths.
proto *proto*	In addition to the special protocols listed, the contributing protocol may be chosen from among any of the ones supported (and currently configured into) **gated**.
as *autonomous_system*	Restrict selection of routes to those learned from the specified autonomous system.
tag *tag*	Restrict selection of routes to those with the specified tag.
aspath *aspath_regexp*	Restrict selection of routes to those that match the specified AS path.
restrict	Indicates that these routes are not to be considered as contributors of the specified aggregate. The specified protocol may be any of the protocols supported by the **gated** daemon.
route_filter	See the section on Route Filters for more detail.

A route may only contribute to an aggregate route that is more general than itself; it must match the aggregate under its mask. Any given route may only contribute to one aggregate route, which will be the most specific configured, but an aggregate route may contribute to a more general aggregate.

Route Filters

All the formats allow route filters as shown below. See the section on route filters for a detailed explanation of how they work. When no route filtering is specified (that is, when `restrict` is specified on the first line of a statement), all routes from the specified source will match that statement. If any filters are specified, only routes that match the specified filters will be considered as contributors. Put differently, if any filters are specified, an `all restrict ;` is assumed at the end of the list.

```
network [ exact | refines ]
network mask mask [exact | refines ]
network masklen number [ exact | refines ]
default
host host
```

Preference

Preference is the value the **gated** daemon uses to order preference of routes from one protocol or peer over another. Preference can be set in the **gated.conf** configuration file in several different configuration statements.

Preference can be set based on network interface over another, from one protocol over another, or from one remote gateway over another.

Preference may not be used to control the selection of routes within an **IGP**, this is accomplished automatically by the protocol based on metric. Preference may be used to select routes from the same **EGP** learned from different peers or autonomous systems.

Each route has only one preference value associated with it, even though preference can be set at many places in the configuration file. Simply, the last or most specific preference value set for a route is the value used. The preference value is an arbitrarily assigned value used to determine the order of routes to the same destination in a single routing database. The active route is chosen by the lowest preference value.

Some protocols implement a second preference (preference2), sometimes refered to as a tie-breaker.

Selecting a Route

- The route with the best (numerically smallest) preference is preferred.
- If the two routes have the same preference, the route with the best (numerically smallest) preference2 (also known as a tie-breaker) is preferred.
- A route learned from a **IGP** is preferred to a route learned from an **EGP**. Least preferred is a route learned indirectly by an **IGP** from an **EGP**.
- If AS path information is available it is used to help determine the most preferred route.
  - A route with an AS path is preferred over one without an AS path.
  - If the AS paths and origins are identical, the route with the lower metric is preferred.
  - A route with an AS path origin of **IGP** is preferred over a route with an AS path origin of **EGP**. Least preferred is an AS path with an **unknown** origin.
  - A route with a shorter AS path is preferred.
- If both routes are from the same protocol and AS, the one with the lowest metric is preferred.
- The route with the lowest numeric next-hop address is used.

Assigning Preferences

A default preference is assigned to each source from which the **gated** daemon receives routes. Preference values range from 0 to 255 with the lowest number indicating the most preferred route.

The following table summarizes the default preference values for routes learned in various ways. The table lists the statements (some of these are clauses within statements) that set preference, and shows the types of routes to which each statement applies. The default preference for each type of route is listed, and the table notes preference precedence between protocols. The narrower the scope of the statement, the higher precedence its preference value is given, but the smaller the set of routes it affects.

Preference Of	**Defined by Statement**	**Default**
direct connnected networks	interface	0
OSPF routes	ospf	10
IS-IS level 1 routes	isis level 1	15
IS-IS level 2 routes	isis level 2	18
internally generated default	gendefault	20
redirects	redirect	30
routes learned via route socket	kernel	40
static routes from config	static	60
ANS SPF (SLSP) routes	slsp	70
HELLO routes	hello	90
RIP routes	rip	100

```
point-to-point interface                                           110
routes to interfaces that are down   interfaces                    120
aggregate/generate routes            aggregate/generate            130
OSPF AS external routes              ospf                          150
BGP routes                           bgp                           170
EGP                                  EGP                           200
```

Sample Preference Specifications

```
interfaces {
        interface 138.66.12.2 preference 10 ;
} ;
rip yes {
    preference 90 ;
} ;
import proto rip gateway 138.66.12.1 preference 75 ;
```

In these statements, the preference applicable to routes learned via RIP from gateway 138.66.12.1 is 75. The last preference applicable to routes learned via RIP from gateway 128.66.12.1 is defined in the accept statement. The preference applicable to other RIP routes is found in the rip statement. The preference set on the interface statement applies only to the route to that interface.

The Router Discovery Protocol

The Router Discovery Protocol is an IETF standard protocol used to inform hosts of the existence of routers. It is used in place of, or in addition to statically configured default routes in hosts.

The protocol is split into two portions, the *server* portion which runs on routers, and the *client* portion that runs on hosts. The **gated** daemon treats these much like two separate protocols, only one of which may be enabled at a time.

The Router Discovery Server

The Router Discovery Server runs on routers and announces their existence to hosts. It does this by periodically multicasting or broadcasting a **Router Advertisement** to each interface on which it is enabled. These Router Advertisements contain a list of all the routers addresses on a given interface and their preference for use as default routers.

Initially, these Router Advertisements occur every few seconds, then fall back to every few minutes. In addition, a host may send a **Router Solicitation** to which the router will respond with a unicast Router Advertisement (unless a multicast or broadcast advertisement is due momentarily).

Each Router Advertisement contains an *Advertisement Lifetime* field indicating for how long the advertised addresses are valid. This lifetime is configured such that another Router Advertisement will be sent before the lifetime has expired. A lifetime of zero is used to indicate that one or more addresses are no longer valid.

The Router Advertisements are by default sent to the all-hosts multicast address `224.0.0.1`. However, the use of broadcast may be specified. When Router Advertisements are being sent to the all-hosts multicast address, or an interface is configured for the limited-broadcast address `255.255.255.255`, all IP addresses configured on the physical interface are included in the Router Advertisement. When the Router Advertisements are being sent to a net or subnet broadcast, only the address associated with that net or subnet is included.

The Router Discovery Server Statement

```
routerdiscovery server yes | no | on | off [ {
    traceoptions trace_options ;
    interface interface_list
        [minadvinterval time]
        [maxadvinterval time]
        [lifetime time]
        ;
    address interface_list
        [advertise] | [ignore]
        [broadcast] | [multicast]
        [ineligible] | [preference preference]
        ;
} ] ;
```

traceoptions *trace_options*

Specifies the Router Discovery tracing options. (See Trace Statements and the Router Discovery specific tracing options below.)

interface *interface_list*

Specifies the parameters that apply to physical interfaces. Note a slight difference in convention from the rest of the gated daemon, **interface** specifies just physical interfaces (such as `en0` and `tr0`), while **address** specifies protocol (in this case IP) addresses.

Interface parameters are:

maxadvinterval *time*

The maximum time allowed between sending broadcast or multicast Router Advertisements from the interface. Must be no less than *4* and no more than *30:00* (30 minutes or 1800 seconds). The default is **10:00** (10 minutes or 600 seconds).

minadvinterval *time*

The minimum time allowed between sending unsolicited broadcast or multicast Router Advertisements from the interface. Must be no less than *3* seconds and no greater than **maxadvinterval**. The default is **0.75 * maxadvinterval**.

lifetime *time*

The lifetime of addresses in a Router Advertisement. Must be no less than **maxadvinterval** and no greater than *2:30:00* (two hours, thirty minutes or 9000 seconds). The default is **3 * maxadvinterval**.

address *interface_list*

Specifies the parameters that apply to the specified set of addresses on this physical interface. Note a slight difference in convention from the rest of **gated.conf**; **interface** specifies just physical interfaces (such as `en0` and `tr0`), while **address** specifies protocol (in this case IP) addresses.

advertise

Specifies that the specified address(es) should be included in Router Advertisements. This is the default.

ignore

Specifies that the specified address(es) should not be included in Router Advertisements.

broadcast

Specifies that the given address(es) should be included in a broadcast Router Advertisement because this system does not support IP multicasting, or some hosts on attached network do not support IP multicasting. It is possible to mix addresses on a physical interface such that some are included in a broadcast Router Advertisement and some are included in a multicast

Router Advertisement. This is the default if the router does not support IP multicasting.

multicast

Specifies that the given address(es) should only be included in a multicast Router Advertisement. If the system does not support IP multicasting the address(es) will not be included. If the system supports IP multicasting, the default is to include the address(es) in a multicast Router Advertisement if the given interface supports IP multicasting, if not the address(es) will be included in a broadcast Router Advertisement.

preference *preference*

The preferability of the address(es) as a default router address, relative to other router addresses on the same subnet. A 32-bit, signed, twos-complement integer, with higher values meaning more preferable. Note that `hex 80000000` may only be specified as `ineligible`. The default is **0**.

ineligible

Specifies that the given address(es) will be assigned a preference of (hex 80000000) that means that it is not eligible to be the default route for any hosts.

This is useful when the address(es) should not be used as a default route, but are given as the next hop in an ICMP redirect. This allows the hosts to verify that the given addresses are up and available.

The Router Discovery Client

A host listens for Router Advertisements via the all-hosts multicast address (`224.0.0.2`), If IP multicasting is available and enabled, or on the interface's broadcast address. When starting up, or when reconfigured, a host may send a few Router Solicitations to the all-routers multicast address, `224.0.0.2`, or the interface's broadcast address.

When a Router Advertisement with non-zero lifetime is received, the host installs a default route to each of the advertised addresses. If the preference **ineligible**, or the address is not on an attached interface, the route is marked unusable but retained. If the preference is usable, the metric is set as a function of the preference such that the route with the best preference is used. If more than one address with the same preference is received, the one with the lowest IP address will be used. These default routes are not exportable to other protocols.

When a Router Advertisement with a zero lifetime is received, the host deletes all routes with next-hop addresses learned from that router. In addition, any routers learned from ICMP redirects pointing to these addresses will be deleted. The same will happen when a Router Advertisement is not received to refresh these routes before the lifetime expires.

The Router Discovery Client Statement

```
routerdiscovery client yes | no | on | off [ {
    traceoptions trace_options ;
    preference preference ;
    interface interface_list
        [ enable ] | [ disable ]
        [ broadcast ] | [ multicast ]
        [ quiet ] | [ solicit ]
        ;
} ] ;
```

traceoptions *trace_options*
Specifies the tracing options for Router Discovery Client. (See Trace Statements and the Router Discovery Client specific tracing options below.)

preference *preference* **;**
Specifies the preference of all Router Discovery default routes. The default is **55**.

interface *interface_list*
Specifies the parameters that apply to physical interfaces. Note a slight difference in convention from the rest of **gated**, **interface** specifies just physical interfaces (such as `en0` and `tr0`). The Router Discovery Client has no parameters that apply only to interface addresses.

enable
Specifies that Router Discovery should be performed on the specified interface(s). This is the default.

disable
Specifies that Router Discovery should not be performed on the specified interface(s).

broadcast
Specifies that Router Solicitations should be broadcast on the specified interface(s). This is the default if IP multicast support is not available on this host or interface.

multicast
Specifies that Router Solicitations should be multicast on the specified interface(s). If IP multicast is not available on this host and interface, no solicitation will be performed. The default is to multicast Router Solicitations if the host and interface support it, otherwise Router Solicitations are broadcast.

quiet
Specifies that no Router Solicitations will be sent on this interface, even though Router Discovery will be performed.

solicit
Specifies that initial Router Solicitations will be sent on this interface. This is the default.

Tracing options

The Router Discovery Client and Server support the `state` trace flag that traces various protocol occurrences.

state	State transitions

The Router Discovery Client and Server do not directly support any packet tracing options, tracing of router discovery packets is enabled via the ICMP Statement.

Route Filtering

Routes are filtered by specifying configuration language that will match a certain set of routes by destination, or by destination and mask. Among other places, route filters are used on `martians`, `import` and `export` statements.

The action taken when no match is found is dependent on the context, for instance `import` and `export` route filters assume an `all reject ;` at the end of a list.

A route will match the most specific filter that applies. Specifying more than one filter with the same destination, mask and modifiers will generate an error.

Filtering syntax

```
network [ exact | refines ]
network mask mask [ exact | refines ]
network masklen number [ exact | refines ]
all
default
host host
```

These are all the possible formats for a route filter. Not all of these formats are available in all places, for instance the `host` and `default` formats are not valid for `martians`.

In most cases it is possible to specify additional parameters relevent to the context of the filter. For example, on a `martian` statement it is possible to specify the `allow` keyword, on an `import` statement you can specify a preference, and on a `export` you can specify a metric.

network [**exact** | **refines**]
network **mask** *mask* [**exact** | **refines**]
network **masklen** *number* [**exact** | **refines**]

Matching usually requires both an address and a mask, although the mask is implied in the shorthand forms listed below. These three forms vary in how the mask is specified. In the first form, the mask is implied to be the natural mask of the network. In the second, the mask is explicitly specified. In the third, the mask is specified by the number of contiguous one bits.

If no additional parameters are specified, any destination that falls in the range given by the network and mask is matched, the mask of the destination is ignored. If a *natural* network is specified, the network, any subnets, and any hosts will be match. The two optional modifiers cause the mask of the destination to be considered also:

exact
This parameter specifies that the mask of the destination must match the supplied mask *exactly*. This is used to match a network, but no subnets or hosts of that network.

refines
Specifies that the mask of the destination must be more specified (that is, longer) than the filter mask. This is used to match subnets and/or hosts of a network, but not the network.

all

This entry matches anything. It is equivalent to:

```
0.0.0.0 mask 0.0.0.0
```

default

Matches the **default** route. To match, the address must be the default address and the mask must be all zeros. This is equivalent to:

```
0.0.0.0 mask 0.0.0.0 exact
```

host *host*

Matches the specific host. To match, the address must exactly match the specified *host* and the network mask must be a host mask (that is, all ones). This is equivalent to:

```
host mask 255.255.255 exact
```

Matching AS Paths

An AS path is a list of autonomous_systems that routing information has passed through to get to this router, and an indicator of the origin of the AS path. This information can be used to prefer one path to a destination network over another. The primary method for doing this with **gated.conf** is to specify a list of patterns to be applied to AS paths when `importing` and `exporting` routes.

Each autonomous system that a route passed through prepends its AS number to the beginning of the AS path.

The origin information details the completeness of AS path information. An origin of **IGP** indicates the route was learned from an interior routing protocol and is most likely complete. An origin of **EGP** indicates the route was learned from an exterior routing protocol that does not support AS paths (EGP, for example) and the path is most likely not complete. When the path information is definitely not complete, an origin of **incomplete** is used.

AS path regular expressions are defined in RFC 1164 section 4.2.

AS Path Matching Syntax

An AS path is matched using the following syntax:

```
aspath aspath_regexp origin any | ( [IGP] [EGP] [incomplete] )
```

This specifies that an AS matching the *aspath_regexp* with the specified origin is matched.

AS Path Regular Expressions

Technically, an AS path regular expression is a regular expression with the alphabet being the set of AS numbers. An AS path regular expression is composed of one or more AS paths expressions. An AS path expressions is composed of AS path terms and AS path operators.

AS Path Terms

An AS path term is one of the following three objects:

```
autonomous_system
.
( aspath_regexp )
```

autonomous_system	Is any valid autonomous system number, from one through 65534 inclusive.
.	Matches any autonomous system number.
(*aspath_regexp* **)**	Contains parentheses group subexpressions--an operator, such as `*` or `?` works on a single element or on a regular expression enclosed in parentheses.

AS Path Operators

An AS path operator is one of the following:

```
aspath_term {m,n}
aspath_term {m}
aspath_term {m,}
aspath_term *
aspath_term +
aspath_term ?
aspath_term | aspath_term
```

aspath_term **{m,n}**	a regular expression followed by `{m,n}` (where m and n are both non-negative integers and m <= n) means at least `m` and at most `n` repetitions.
aspath_term **{m}**	a regular expression followed by `{m}` (where m is a positive integer) means exactly `m` repetitions.
aspath_term **{m,}**	a regular expression followed by `{m,}` (where m is a positive integer) means `m or more` repetitions.
aspath_term *****	an AS path term followed by `*` means `zero or more` repetitions. This is shorthand for `{0,}`.
aspath_term **+**	a regular expression followed by `+` means `one or more` repetitions. This is shorthand for `{1,}`.
aspath_term **?**	a regular expression followed by `?` means `zero` or `one` repetition. This is shorthand for `{0,1}`.
aspath_term **|** *aspath_term*	matches the AS term on the left, or the AS term on the right.

gateways File Format for TCP/IP

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. Gateways that the daemon cannot identify through its queries 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 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, the gateway is expected to exchange Routing Information Protocol (RIP) information. As long as the gateway is active, information about it is maintained in the internal routing tables. This information is included with any routing information transmitted through RIP. If the gateway does not respond for a period of time, the associated route is deleted from the internal routing tables.
passive	A passive gateway is not expected to exchange RIP information. Information about the gateway is maintained in the routing tables indefinitely and is included with any routing information 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 instead through an unspecified routing protocol).

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/gateways	Contains the sample **gateways** file, which also contains directions for its use.

Related Information

The **routed** daemon.

Gateways for TCP/IP, TCP/IP Protocols, TCP/IP Routing in

hosts File Format for TCP/IP

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

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 . (period) 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.

> **Note:** Valid host names or alias host names must contain at least one alphabetic character. If you choose to specify a host name or alias that begins with an `x` followed by any hexadecimal digit (`0-f`), the host name or alias must also contain at least one additional letter that cannot be expressed as a hexadecimal digit. The system interprets a leading `x` followed by a hexadecimal digit as the base 16 representation of an address, unless there is at least one character in the host name or alias that is not a hexadecimal digit. Thus, `xdeer` would be a valid host name, whereas `xdee` would not.

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 by using the System Management Interface Tool (SMIT) or 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.century.com  name1
128.114.1.14  name2.xyz.aus.century.com  name2
128.114.1.16  name3.xyz.aus.century.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 Base Operating System (BOS) Runtime.

Related Information

The **hostent** command, **setclock** command.

The **gethostbyaddr** routine.

hosts.equiv File Format for TCP/IP

Purpose

Specifies remote systems that can execute commands on the local system.

Description

The **/etc/hosts.equiv** file, along with any local **$HOME/.rhosts** files, defines the hosts (computers on a network) and user accounts that can invoke remote commands on a local host without supplying a password. A user or host that is not required to supply a password is considered trusted.

When a local host receives a remote command request, the appropriate local daemon first checks the **/etc/hosts.equiv** file to determine if the request originates with a trusted user or host. For example, if the local host receives a remote login request, the **rlogind** daemon checks for the existence of a **hosts.equiv** file on the local host. If the file exists but does not define the host or user, the system checks the appropriate **$HOME/.rhosts** file. This file is similar to the **/etc/hosts.equiv** file, except that it is maintained for individual users.

Both files, **hosts.equiv** and **.rhosts** must have permissions denying write access to group and other. If either group or other have write access to a file, that file will be ignored.

Do not give write permission to the **/etc/hosts.equiv** file to group and others. Permissions of the **/etc/hosts.equiv** file should be set to 600 (read and write by owner only).

If a remote command request is made by the root user, the **/etc/hosts.equiv** file is ignored and only the **/.rhosts** file is read.

Note: Be careful when establishing trusted relationships. Networks that use trusted facilities can be less secure than those that do not.

Granting and Denying Trust

You grant trust from a local host to a remote host or remote user. The local machine's **/etc/hosts.equiv** file contains entries for each trusted host or user. The format of an entry is:

HostName [*UserName*]

The *HostName* field specifies the name of the host to trust. The *UserName* field specifies the name of the user on that remote host to trust. The *UserName* field is optional.

You can use the + (plus sign) as a wildcard in either the *HostName* or *UserName* field to grant trust to all users from a particular host or from all hosts that a specific user has an account on. To grant trust to every user on every machine on the network, place a plus sign (+) at the beginning of the file.

Note: When granting access through the **/etc/hosts.equiv** file, extreme caution must be used. Lines that include a *UserName*, either as an individual user, a netgroup, or the + (plus sign used as a wildcard character), permit the qualifying users to access the system as any non-root local user.

You deny a host or user trust by omitting them from the **/etc/hosts.equiv** file altogether. By omitting the host or user, you imply they are not trusted. This is the most secure way to deny trust. Otherwise, you can explicitly deny trust to a specific host or user by using the **-** (minus sign). The format to explicitly deny a host is:

-*HostName*

The format to explicitly deny a specific user from a host is:

HostName [**-***UserName*]

Using NIS with the /etc/hosts.equiv file

If your network uses the Network Information Services (NIS), you can use netgroups in place of either the *HostName* or *UserName* field. The system resolves the netgroup depending on which field the netgroup replaces. For example, if you place a netgroup in the *HostName* field, the system resolves the hosts component of the netgroup. If the netgroup appears in the *UserName* field, the user component is resolved. Use the following format to grant trust to a netgroup:

+@*NetGroup*

To deny trust, use the following:

-@*NetGroup*

Refer to the NIS **netgroup** file for more information on netgroups.

Ordering Entries in the /etc/hosts.equiv File

The order of entries in the **/etc/hosts.equiv** file is important. When verifying trust, the system parses the **/etc/hosts.equiv** file from top to bottom. When it encounters an entry that matches the host or user attempting a remote command, the system stops parsing the file and grants or denies trust based on the entry. Any additional entries that appear later in the file are ignored.

Examples

1. To allow all the users on remote hosts `emerald` and `amethyst` to log in to host `diamond`, enter:

   ```
   emerald
   amethyst
   ```

   These entries in `diamond`'s **/etc/hosts.equiv** file allow all the users on `emerald` and `amethyst` with local accounts on `diamond` to remotely log in without supplying a password.
2. To allow only the user `gregory` to remotely login to `diamond` from host `amethyst`, enter:

   ```
   emerald
   amethyst gregory
   ```

   This entry in `diamond`'s **/etc/hosts.equiv** file forces all the users on `amethyst`, except for `gregory`, to supply a password when remotely logging in to `diamond`.
3. To grant trust to `peter` regardless of the host he attempts to execute remote commands from,

enter:

```
emerald
amethyst gregory
+ peter
```

This entry in `diamond`'s **/etc/hosts.equiv** file allows `peter` to execute remote commands on `diamond` from any host that he has an account on.

4. To allow all hosts in the `century` netgroup to execute remote commands on host `diamond`, enter:

```
emerald
amethyst gregory
+ peter
+@century
```

This entry in `diamond`'s **/etc/hosts.equiv** file grants trust to all hosts in the `century` netgroup. This means that any user with an account on a `century` host and an account on `diamond` can execute remote commands on `diamond` without supplying a password.

5. To allow all the users in the `engineers` netgroup with accounts on `citrine` to execute remote commands on host `diamond`, enter:

```
emerald
amethyst gregory
+ peter
+@century
citrine +@engineers
```

This entry in `diamond`'s **/etc/hosts.equiv** file grants trust to all of netgroup `engineers` users with an account on `citrine.`

6. To grant trust to all users with accounts on hosts in the `servers` netgroup that are users in the `sysadmins` netgroup, enter:

```
emerald
amethyst gregory
+ peter
+@century
citrine +@engineers
+@servers +@sysadmins
```

This entry in `diamond`'s **/etc/hosts.equiv** file grants trust to any user in the `sysadmins` netgroup who is remotely executing commands from hosts that are in the `servers` netgroup.

7. To force an `engineers` netgroup user `lydia` who has an account on `citrine` to use a password while allowing all other `engineers` users not to, enter:

```
emerald
amethyst gregory
+ peter
+@century
citrine -lydia
citrine +@engineers
+@servers +@sysadmins
```

This entry in `diamond`'s **/etc/hosts.equiv** file grants trust to all of netgroup `engineers` users, except for `lydia`, who must supply a password. The order of entries is very important. Recall that the system grants trust based on the first entry it encounters. If the order of the entries appeared as follows:

```
emerald
amethyst gregory
+ peter
+@century
citrine +@engineers
citrine -lydia
+@servers +@sysadmins
```

User `lydia`, as a member of `engineers`, would be allowed to execute remote commands on `diamond` even though a later entry explicitly denies her trust.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

$HOME/.rhosts	Specifies remote users who can use a local-user account.

Related Information

The NIS **netgroup** file.

The TCP/IP **.rhosts** file format.

The **lpd** command, **rcp** command, **rdist** command, **rdump** command, **rlogin** command, **rsh** command, **ruser** command.

The **rlogind** daemon, **rshd** daemon.

hosts.lpd File Format for TCP/IP

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 do not have the full privileges given to files listed in the **/etc/hosts.equiv** file.

Host-Name Field

The **hosts.lpd** file supports the following host-name entries:

```
+
HostName
-HostName
+@NetGroup
-@NetGroup
```

A + (plus sign) signifies that any host on the network can print using the local host. The `HostName` entry is the name of a remote host and signifies that `HostName` can print, using the local host. A `-HostName` entry signifies the host is not allowed to print using the local host. A `+@NetGroup` or `-@NetGroup` entry signifies all hosts in the netgroup or no hosts in the netgroup, respectively, are allowed to print using the local host.

The *@NetGroup* parameter is used by Network Information Service (NIS) for grouping. Refer to the NIS **netgroup** file for more information on netgroups.

Entries in this file can be made using the System Management Interface Tool (SMIT) or the **ruser** command.

> **Note:** Comments must be entered on separate lines in the **hosts.lpd** file. Comments should not be entered on lines containing host names.

To implement **hosts.lpd** file changes without restarting the system, use the System Resource Controller (SRC) **refresh** command.

Examples

1. To allow remote specified hosts to print using a local host, enter:

   ```
   hamlet
   lear
   prospero
   setebos
   ```

   These entries in the local host's **/etc/hosts.lpd** file allow hosts `hamlet`, `lear`, `prospero`, and `setebos` to print files, using the local host.

2. To prevent a remote host from printing using a local host, enter:

   ```
   -hamlet
   ```

   This entry in the local host's **/etc/hosts.lpd** file prevents host `hamlet` from printing files, using the local host.

3. To allow all hosts in an NIS netgroup to print using the local host, enter:

   ```
   +@century
   ```

   This entry in the local host's **/etc/hosts.lpd** file allows all hosts in the `century` netgroup to print files, using the local host. The @ (at sign) signifies the network is using NIS grouping.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/etc/hosts.equiv	Specifies remote systems that can execute commands on the local system.

Related Information

The **netgroup** file for NIS.

The **hosts.equiv** file format for TCP/IP.

The **lpd** command, **ruser** command.

hty_config File Format

Purpose

Specifies the number of htys to configure on a Network Terminal Accelerator adapter.

Description

The **/etc/hty_config** file supplies the **hty_load** command with information to define ports for a specified device. The System Management Interface Tool (SMIT) writes to this file when **hty** devices are configured, specifying the device by supplying the adapter minor number for the device. Both the number of ports and the device are specified in a three-column table that can have multiple lines.

The `Cluster Address` column defines the cluster controller's network address. For the boards, the cluster address should be set to 1. Any other value may cause unpredictable results.

After you have configured the Network Terminal Accelerator adapter with SMIT, the **hty_config** file appears similar to the following:

```
Adapter       Cluster         Number
minor #       address         of ports
-------       -------         --------
0               1               256
1               1               700
2               1               85
```

In this example, the host has three adapters, the first of which is configured for 256 **hty** devices, the second for 700, and the third for 85.

See "Configuring the Network Terminal Accelerator," in *AIX Versions 3.2 and 4 Asynchronous Communications Guide* for more information on configuring the adapter.

Related Information

inetd.conf File Format for TCP/IP

Purpose

Defines how the **inetd** daemon handles Internet service requests.

Description

The **/etc/inetd.conf** file is the default configuration file for the **inetd** daemon. This file enables you to specify the daemons to start by default and supply the arguments that correspond to the desired style of functioning for each daemon.

If you change the **/etc/inetd.conf** file, run the **refresh -s inetd** or **kill -1** *InetdPID* command to inform the **inetd** daemon of the changes to its configuration file. The **inetd.conf** file specifies which daemons start by default and supplies arguments determining the style of functioning for each daemon.

The following daemons are controlled by the **inetd** daemon:

- **comsat**
- **ftpd**
- **telnetd**
- **rshd**
- **rlogind**
- **rexecd**
- **fingerd**
- **tftpd**
- **talkd**
- **uucpd**

The **ftpd**, **rlogind**, **rexecd**, **rshd**, **talkd**, **telnetd**, and **uucpd** daemons are started by default. The **tftpd**, **fingerd**, and **comsat** daemons are not started by default unless they are uncommented in the **/etc/inetd.conf** file.

Service Requests

The following Internet service requests are supported internally by the **inetd** daemon and are generally used for debugging:

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 user-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 only.

Each line is of the form:

ServiceName SocketType ProtocolName Wait/NoWait UserName ServerPath ServerArgs

These fields must be separated by spaces or tabs and 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. Possible values for the *SocketType* parameter are:

  stream Specifies that a stream socket is used for the service.

  dgram Specifies that a datagram socket is used for the service

  sunrpc_tcp Specifies that a Sun remote procedure call (RPC) socket is used for the service, over a stream connection.

  sunrpc_udp Specifies that a Sun RPC socket is used for the service, over a datagram connection.

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 TCP/IP and the **udp** value for a service that uses the User Datagram Protocol (UDP).

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.

Wait/NoWait/SRC
: Contains either the **wait**, the **nowait**, or the **SRC** instruction for datagram sockets and the **nowait** instruction for stream sockets. The *Wait/NoWait/SRC* field determines whether the **inetd** daemon waits for a datagram server to release the socket before continuing to listen at the socket. The **SRC** instruction works like wait, but instead of forking and waiting for the child to die, it does a **startsrc** on the subsystem and stores information about the starting of the service. When the service is removed from the **inetd.conf** file and **inetd** is restarted, the service then has a **stopsrc** issued to the service to stop it.

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. The maximum number of arguments is five. The first argument specifies the name of the server used. If the *SocketType* parameter is **sunrpc_tcp** or **sunrpc_udp**, the second argument specifies the program name and the third argument specifies the version of the program. For services that the **inetd** daemon provides internally, this field should be empty.

Examples

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 the **talkd** daemon for **ntalk** requests
- Provides time requests internally.

```
ftp stream tcp nowait root /usr/sbin/ftpd ftpd
ntalk dgram udp wait root /usr/sbin/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 Base Operating System (BOS) Runtime.

Files

etc/services	Defines the sockets and protocols used for Internet services.
/etc/protocols	Defines the Internet protocols used on the local host.

Related Information

The **kill** command, **refresh** command.

The **inetd** daemon.

The **protocols** file format, **services** file format.

lastlog File Format

Purpose

Defines the last login attributes for users.

Description

The **/etc/security/lastlog** file is an ASCII file that contains stanzas with the last login attributes for users. Each stanza is identified by a user name 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.

Each stanza can have the following attributes:

time_last_login	Specifies the number of seconds since the epoch (00:00:00 GMT, January 1, 1970) since the last successful login. The value is a decimal integer.
tty_last_login	Specifies the terminal on which the user last logged in. The value is a character string.
host_last_login	Specifies the host from which the user last logged in. The value is a character string.
unsuccessful_login_count	Specifies the number of unsuccessful login attempts since the last successful login. The value is a decimal integer. This attribute works in conjunction with the user's loginretries attribute, specified in the **/etc/security/user** file, to lock the user's account after a specified number of consecutive unsuccessful login attempts. Once the user's account is locked, the user will not be able to log in until the system administrator resets the user's unsuccessful_login_count attribute to be less than the value of loginretries. To do this, enter the following: `chsec -f /etc/security/lastlog -s username -a \ unsuccessful_login_count=0`
time_last_unsuccessful_login	Specifies the number of seconds since the epoch (00:00:00 GMT, January 1, 1970) since the last unsuccessful login. The value is a decimal integer.
tty_last_unsuccessful_login	Specifies the terminal on which the last unsuccessful login attempt occurred. The value is a character string.
host_last_unsuccessful_login	Specifies the host from which the last unsuccessful login attempt occurred. The value is a character string.

All user database files should be accessed 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 the **lastlog** file. The attributes of this user stanza are initially empty. The field values are set by the **login** command as a result of logging in to the system. The **lsuser** command displays the values of these attributes; the **rmuser** command removes the user stanza from this file, along with the user account.

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.

Examples

A typical stanza is similar to the following example for user `bck`:

```
bck:
   time_last_unsuccessful_login = 732475345
   tty_last_unsuccessful_login = tty0
   host_last_unsuccessful_login = waterski
   unsuccessful_login_count = 0
   time_last_login = 734718467
   tty_last_login = lft/0
   host_last_login = waterski
```

Implementation Specifics

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

Files

/etc/security/lastlog	Specifies the path to the **lastlog** 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/environ	Contains the environment attributes of users.
/etc/security/user	Contains the extended attributes of users.
/etc/security/limits	Contains the process resource limits of users.

Related Information

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

Locale Definition Source File Format

Purpose

Contains one or more categories that describe a locale.

Description

A locale definition source file contains one or more categories that describe a locale. Files using this format can be converted into a locale by using the **localedef** command. Locales can be modified only by editing a locale definition source file and then using the **localedef** command again on the new source file. Locales are not affected by a locale definition source file unless the file is first converted using the **localedef** command.

The locale definition source file sections define categories of locale data. A source file should not contain more than one section for the same category. The following categories are supported:

LC_COLLATE	Defines character or string collation information.
LC_CTYPE	Defines character classification, case conversion, and other character attributes.
LC_MESSAGES	Defines the format for affirmative and negative responses.
LC_MONETARY	Defines rules and symbols for formatting monetary numeric information.
LC_NUMERIC	Defines a list of rules and symbols for formatting non-monetary numeric information.
LC_TIME	Defines a list of rules and symbols for formatting time and date information.

The category definition consists of:

- The category header (category name)
- The associated keyword/value pairs that comprise the category body
- The category trailer (which consists of `END` *category-name*)

For example:

LC_CTYPE
source for LC_CTYPE category
END LC_CTYPE

The source for all of the categories is specified using keywords, strings, character literals, and character symbols. Each keyword identifies either a definition or a rule. The remainder of the statement containing the keyword contains the operands to the keyword. Operands are separated from the keyword by one or more blank characters. A statement may be continued on the next line by placing a / (slash) as the last character before the new-line character that terminates the line. Lines containing the *comment_char* entry in the first column are treated as comment lines. The default is # (pound sign).

The first category header in the file can be preceded by a line that changes the comment character. It has the following format, starting in column 1:

```
comment_char character
```

where *character* is the new comment character.

Blank lines and lines containing the comment character in the first position are ignored.

A character symbol begins with the < (less-than) character, followed by up to 30 non-control, non-space characters, and ends with the > (greater-than) character. For example, `<A-diaeresis>` is a valid character symbol. Any character symbol referenced in the source file should either be one of the portable character set symbols or should be defined in the provided character set description (**charmap**) source file.

A character literal is the character itself, or else a decimal, hexadecimal, or octal constant. A decimal constant is of the form:

```
\dxxx
```

where `x` is a decimal digit. A hexadecimal constant is of the form:

```
\xddd
```

where `d` is a hexadecimal digit. An octal constant is of the form:

```
\ddd
```

where `d` is an octal digit.

A string is a sequence of character symbols, or literals enclosed by " " (double-quotation marks). For example:

```
"<A-diaeresis> \d65\d120 <B>"
```

The explicit definition of each category in a locale definition source file is not required. When a category is undefined in a locale definition source file, it defaults to the C locale definition.

The first category header in the file can be preceded by a line that changes the escape character used in the file. It has the following format, starting in column 1:

```
escape_char character
```

where *character* is the new escape character.

The escape character defaults to the / (backslash).

Implementation Specifics

This source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **locale** command, **localedef** command.

Character Set Description (charmap) Source File Format , Locale Method Source File Format .

For specific information about the locale categories and their keywords, see the **LC_COLLATE** category, **LC_CTYPE** category, **LC_MESSAGES** category, **LC_MONETARY** category, **LC_NUMERIC** category, and **LC_TIME** category for the locale definition source file format.

Changing Your Locale in *AIX Version 4.3 System Management Guide: Operating System and Devices*.

LC_COLLATE Category for the Locale Definition Source File Format

Purpose

Defines character or string collation information.

Description

A collation element is the unit of comparison for collation. A collation element may be a character or a sequence of characters. Every collation element in the locale has a set of weights, which determine if the collation element collates before, equal to, or after the other collation elements in the locale. Each collation element is assigned collation weights by the **localedef** command when the locale definition source file is converted. These collation weights are then used by applications programs that compare strings.

Comparison of strings is performed by comparing the collation weights of each character in the string until either a difference is found or the strings are determined to be equal. This comparison may be performed several times if the locale defines multiple collation orders. For example, in the French locale, the strings are compared using a primary set of collation weights. If they are equal on the basis of this comparison, they are compared again using a secondary set of collation weights. A collating element has a set of collation weights associated with it that is equal to the number of collation orders defined for the locale.

Every character defined in the **charmap** file (or every character in the portable character set if no **charmap** file is specified) is itself a collating element. Additional collating elements can be defined using the **collating-element** statement. The syntax is:

collating-element *character-symbol* **from** *string*

The **LC_COLLATE** category begins with the **LC_COLLATE** keyword and ends with the **END LC_COLLATE** keyword.

The following keywords are recognized in the **LC_COLLATE** category:

copy	The **copy** statement specifies the name of an existing locale to be used as the definition of this category. If a **copy** statement is included in the file, no other keyword can be specified.
collating-element	The **collating-element** statement specifies multicharacter collating elements.

The syntax for the **collating-element** statement is:

collating-element <*collating-symbol*> **from** <*string*>

The *collating-symbol* value defines a collating element that is a string of one or more characters as a single collating element. The *collating-symbol* value cannot duplicate any symbolic name in the current **charmap** file, or any other symbolic name defined in this collation definition. The *string* value specifies a string of two or more characters that define the *collating-symbol* value. Following are examples of the syntax for the **collating-element** statement:

collating-element <*ch***> from <***c***><***h***>**
collating-element <*e-acute***> from <***acute***><***e***>**
collating-element <*11***> from <***1***><***1***>**

A *collating-symbol* value defined by the **collating-element** statement is recognized only with the **LC_COLLATE** category.

collating-symbol The **collating-symbol** statement specifies collation symbols for use in collation sequence statements.

The syntax for the **collating-symbol** statement is:

collating-symbol <*collating-symbol***>**

The *collating-symbol* value cannot duplicate any symbolic name in the current **charmap** file, or any other symbolic name defined in this collation definition. Following are examples of the syntax for the **collating-symbol** statement:

collating-symbol <*UPPER_CASE***>**
collating-symbol <*HIGH***>**

A *collating-symbol* value defined by the **collating-symbol** statement is recognized only within the **LC_COLLATE** category.

order_start The **order_start** statement must be followed by one or more collation order statements, assigning collation weights to collating elements. This statement is mandatory.

The syntax for the **order_start** statement is:

order_start <*sort-rules*>**,** <*sort-rules*>**,**...<*sort-rules*>
collation order statements
order_end

The <*sort-rules*> directives have the following syntax:

keyword, *keyword*,...*keyword*; *keyword*, *keyword*,...*keyword*

where *keyword* is one of the keywords **forward**, **backward**, and **position**.

The *sort-rules* directives are optional. If present, they define the rules to apply during string comparison. The number of specified *sort-rules* directives defines the number of weights each collating element is assigned (that is, the number of collation orders in the locale). If no *sort-rules* directives are present, one **forward** keyword is assumed and comparisons are made on a character basis rather than a string basis. If present, the first *sort-rules* directive applies when comparing strings using primary weight, the second when comparing strings using the secondary weight, and so on. Each set of *sort-rules* directives is separated by a ; (semicolon). A *sort-rules* directive consists of one or more comma-separated keywords. The following keywords are supported:

forward Specifies that collation weight comparisons proceed from the beginning of a string toward the end of the string.

backward Specifies that collation weight comparisons proceed from the end of a string toward the beginning of the string.

position Specifies that collation weight comparisons consider the relative position of elements in the string not subject to the special symbol **IGNORE**. That is, if strings compare equal, the element with the shortest distance from the starting point of the string collates first.

The **forward** and **backward** keywords are mutually exclusive. Following is an example of the syntax for the <*sort-rules*> directives:

order_start *forward; backward, position*

The optional operands for each collation element are used to define the primary, secondary, or subsequent weights for the collating element. The special symbol **IGNORE** is used to indicate a collating element that is to be ignored when strings are compared.

A collation statement with the **ellipsis** keyword on the left-hand side results in the *collating-element-list* on the right-hand side being applied to every character with an encoding that falls numerically between the character on the left-hand side in the preceding statement and the character on the left-hand side of the following statement. If the **ellipsis** occur in the first statement, it is interpreted as though the preceding line specified the **NUL** character. (The **NUL** character is a character with all bits set to 0.) If the **ellipsis** occur in the last statement, it is interpreted as though the following line specified the greatest encoded value.

An **ellipsis** keyword appearing in place of a *collating-element-list* indicates the weights are to be assigned, for the characters in the identified range, in numerically increasing order from the weight for the character symbol on the left-hand side of the preceding statement.

> **Note:** The use of the **ellipsis** keyword results in a locale that may collate differently when compiled with different character set description (**charmap**) source files. For this reason, the **localedef** command issues a warning when the **ellipsis** keyword is encountered.

All characters in the character set must be placed in the collation order, either explicitly or implicitly by using the **UNDEFINED** special symbol. The **UNDEFINED** special symbol includes all coded character set values not specified explicitly or with an ellipsis symbol. These characters are inserted in the character collation order at the point indicated by the **UNDEFINED** special symbol in the order of their character code set values. If no **UNDEFINED** special symbol exists and the collation order does not specify all collation elements from the coded character set, a warning is issued and all undefined characters are placed at the end of the character collation order.

Examples

The following is an example of a collation order statement in the **LC_COLLATE** locale definition source file category:

```
order_start     forward;backward
UNDEFINED       IGNORE;IGNORE
<LOW>           <LOW>;<space>
...             <LOW>;...
<a>             <a>;<a>
<a-acute>       <a>;<a-acute>
<a-grave>       <a>;<a-grave>
<A>             <a>;<A>
<A-acute>       <a>;<A-acute>
<A-grave>       <a>;<A-grave>
<ch>            <ch>;<ch>
<Ch>            <ch>;<Ch>
<s>             <s>;<s>
<ss>            <s><s>;<s><s>
<eszet> <s><s>;<eszet><eszet>
...             <HIGH>;...
<HIGH>
order_end
```

This example is interpreted as follows:

- The **UNDEFINED** special symbol indicates that all characters not specified in the definition (either explicitly or by the ellipsis symbol) are ignored for collation purposes.
- All collating elements between <space> and <a> have the same primary equivalence class and individual secondary weights based on their coded character set values.
- All characters based on the uppercase or lowercase a character belong to the same primary equivalence class.
- The <c><h> multicharacter collating element is represented by the <ch> collating symbol and belongs to the same primary equivalence class as the <C><h> multicharacter collating element.
- The <eszet> character is collated as an <s><s> string. That is, one <eszet> character is expanded to two characters before comparing.

Implementation Specifics

This category of the locale definition source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **ed** command, **locale** command, **localedef** command.

Character Set Description (charmap) Source File Format , Locale Definition Source File Format , Locale Method Source File Format .

For specific information about other locale categories and their keywords, see the **LC_CTYPE** category, **LC_MESSAGES** category, **LC_MONETARY** category, **LC_NUMERIC** category, and **LC_TIME** category for the locale definition source file format.

LC_CTYPE Category for the Locale Definition Source File Format

Purpose

Defines character classification, case conversion, and other character attributes.

Description

The **LC_CTYPE** category of a locale definition source file defines character classification, case conversion, and other character attributes. This category begins with an **LC_CTYPE** category header and terminates with an **END LC_CTYPE** category trailer.

All operands for **LC_CTYPE** category statements are defined as lists of characters. Each list consists of one or more semicolon-separated characters or symbolic character names.

The following keywords are recognized in the **LC_CTYPE** category. In the descriptions, the term *automatically included* means that an error does not occur if the referenced characters are included or omitted. The characters will be provided if they are missing and will be accepted if they are present.

copy	Specifies the name of an existing locale to be used as the definition of this category. If a **copy** statement is included in the file, no other keyword can be specified.
upper	Defines uppercase letter characters. No character defined by the **cntrl**, **digit**, **punct**, or **space** keyword can be specified. At a minimum, the uppercase letters A-Z must be defined.
lower	Defines lowercase letter characters. No character defined by the **cntrl**, **digit**, **punct**, or **space** keyword can be specified. At a minimum, the lowercase letters a-z must be defined.
alpha	Defines all letter characters. No character defined by the **cntrl**, **digit**, **punct**, or **space** keyword can be specified. Characters defined by the **upper** and **lower** keywords are automatically included in this character class.
digit	Defines numeric digit characters. Only the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 can be specified.
alnum	Defines alphanumeric characters. No character defined by the **cntrl**, **punct**, or **space** keyword can be specified. Characters defined by the **alpha** and **digit** keywords are automatically included in this character class.

space
Defines whitespace characters. No character defined by the **upper**, **lower**, **alpha**, **digit**, **graph**, **cntrl**, or **xdigit** keyword can be specified. At a minimum, the <space>, <form-feed>, <newline>, <carriage return>, <tab>, and <vertical-tab> characters, and any characters defined by the **blank** keyword, must be specified.

cntrl
Defines control characters. No character defined by the **upper**, **lower**, **alpha**, **digit**, **punct**, **graph**, **print**, **xdigit**, or **space** keyword can be specified.

punct
Defines punctuation characters. A character defined as the <space> character and characters defined by the **upper**, **lower**, **alpha**, **digit**, **cntrl**, or **xdigit** keyword cannot be specified.

graph
Defines printable characters, excluding the <space> character. If this keyword is not specified, characters defined by the **upper**, **lower**, **alpha**, **digit**, **xdigit**, and **punct** keywords are automatically included in this character class. No character defined by the **cntrl** keyword can be specified.

print
Defines printable characters, including the <space> character. If this keyword is not specified, the <space> character and characters defined by the **upper**, **lower**, **alpha**, **digit**, **xdigit**, and **punct** keywords are automatically included in this character class. No character defined by the **cntrl** keyword can be specified.

xdigit
Defines hexadecimal digit characters. The digits 0-9 and the letters A-F and a-f can be specified. The **xdigit** keyword defaults to its normal class limits.

blank
Defines blank characters. If this keyword is not specified, the <space> and <horizontal-tab> characters are included in this character class. Any characters defined by this statement are automatically included in the **space** keyword class.

charclass
Defines one or more locale-specific character class names as strings separated by semicolons. Each named character class can then be defined subsequently in the **LC_CTYPE** definition. A character class name consists of at least one, and at most 32 bytes, of alphanumeric characters from the portable character set symbols. The first character of a character class name cannot be a digit. The name cannot match any of the **LC_CTYPE** keywords defined in this section.

charclass-name
Defines characters to be classified as belonging to the named locale-specific character class. Locale-specific named character classes need not exist in the POSIX locale.

If a class name is defined by a **charclass** keyword, but no characters are subsequently assigned to it, it represents a class without any characters belonging to it.

The *charclass-name* can be used as the *Property* parameter in the **wctype** subroutine, in regular expressions and shell pattern-matching expressions, and by the **tr** command.

toupper Defines the mapping of lowercase characters to uppercase characters. Operands for this keyword consist of semicolon-separated character pairs. Each character pair is enclosed in ( ) (parentheses) and separated from the next pair by a , (comma). The first character in each pair is considered lowercase; the second character is considered uppercase. Only characters defined by the **lower** and **upper** keywords can be specified.

tolower Defines the mapping of uppercase characters to lowercase characters. Operands for this keyword consist of semicolon-separated character pairs. Each character pair is enclosed in ( ) (parentheses) and separated from the next pair by a , (comma). The first character in each pair is considered uppercase; the second character is considered lowercase. Only characters defined by the **lower** and **upper** keywords can be specified.

The **tolower** keyword is optional. If this keyword is not specified, the mapping defaults to the reverse mapping of the **toupper** keyword, if specified. If the **toupper** and **tolower** keywords are both unspecified, the mapping for each defaults to that of the **C** locale.

The **LC_CTYPE** category does not support multicharacter elements. For example, the German sharp-s character is traditionally classified as a lowercase letter. There is no corresponding uppercase letter; in proper capitalization of German text, the sharp-s character is replaced by the two characters ss. This kind of conversion is outside of the scope of the **toupper** and **tolower** keywords.

Examples

The following is an example of a possible **LC_CTYPE** category listed in a locale definition source file:

```
LC_CTYPE
#"alpha" is by default "upper" and "lower"
#"alnum" is by default "alpha" and "digit"
#"print" is by default "alnum", "punct" and the space character
#"graph" is by default "alnum" and "punct"
#"tolower" is by default the reverse mapping of "toupper"
#
upper           <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
                <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
#
lower           <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
                <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
#
digit           <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
                <seven>;<eight>;<nine>
#
space           <tab>;<newline>;<vertical-tab>;<form-feed>;\
                <carriage-return>;<space>
#
cntrl           <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;/
                <form-feed>;<carriage-return>;<NUL>;<SOH>;<STX>;/
                <ETX>;<EOT>;<ENQ>;<ACK>;<SO>;<SI>;<DLE>;<DC1>;<DC2>;/
                <DC3>;<DC4>;<NAK>;<SYN>;<ETB>;<CAN>;<EM>;<SUB>;/
                <ESC>;<IS4>;<IS3>;<IS2>;<IS1>;<DEL>
#
punct           <exclamation-mark>;<quotation-mark>;<number-sign>;\
                <dollar-sign>;<percent-sign>;<ampersand>;<asterisk>;\
                <apostrophe>;<left-parenthesis>;<right-parenthesis>;
```

```
                <plus-sign>;<comma>;<hyphen>;<period>;<slash>;/
                <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
                <greater-than-sign>;<question-mark>;<commercial-at>;\
                <left-square-bracket>;<backslash>;<circumflex>;\
                <right-square-bracket>;<underline>;<grave-accent>;\
                <left-curly-bracket>;<vertical-line>;<tilde>;\
                <right-curly-bracket>
#
xdigit          <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
                <seven>;<eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;\
                <a>;<b>;<c>;<d>;<e>;<f>
#
blank           <space>;<tab>
#
toupper  (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
                (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
                (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
                (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
                (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);\
                (<z>,<Z>)
#
END LC_CTYPE
```

Implementation Specifics

This category of the locale definition source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **locale** command, **localedef** command, **tr** command.

The **wctype** subroutine.

Character Set Description (charmap) Source File Format , Locale Definition Source File Format , Locale Method Source File Format .

For specific information about other locale categories and their keywords, see the **LC_COLLATE** category, **LC_MESSAGES** category, **LC_MONETARY** category, **LC_NUMERIC** category, and **LC_TIME** category for the locale definition source file format.

LC_MESSAGES Category for the Locale Definition Source File Format

Purpose

Defines the format for affirmative and negative system responses.

Description

The **LC_MESSAGES** category of a locale definition source file defines the format for affirmative and negative system responses. This category begins with an **LC_MESSAGES** category header and terminates with an **END LC_MESSAGES** category trailer.

All operands for the **LC_MESSAGES** category are defined as strings or extended regular expressions enclosed by " " (double-quotation marks). These operands are separated from the keyword they define by one or more blanks. Two adjacent " " (double-quotation marks) indicate an undefined value. The following keywords are recognized in the **LC_MESSAGES** category:

copy
: Specifies the name of an existing locale to be used as the definition of this category. If a **copy** statement is included in the file, no other keyword can be specified.

yesexpr
: Specifies an extended regular expression that describes the acceptable affirmative response to a question expecting an affirmative or negative response.

noexpr
: Specifies an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.

yesstr
: A colon-separated string of acceptable affirmative responses. This string is accessible to applications through the **nl_langinfo** subroutine as **nl_langinfo (***YESSTR***)**.

nostr
: A colon-separated string of acceptable negative responses. This string is accessible to applications through the **nl_langinfo** subroutine as **nl_langinfo (***NOSTR***)**.

Examples

The following is an example of a possible **LC_MESSAGES** category listed in a locale definition source file:

```
LC_MESSAGES
#
yesexpr "([yY][[:alpha:]]*)|(OK)"
noexpr          "[nN][[:alpha:]]*"
yesstr          "Y:y:yes"
nostr           "N:n:no"
#
END LC_MESSAGES
```

Implementation Specifics

This category of the locale definition source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **locale** command, **localedef** command.

Character Set Description (charmap) Source File Format , Locale Definition Source File Format , Locale Method Source File Format .

For specific information about other locale categories and their keywords, see the **LC_COLLATE** category, **LC_CTYPE** category, **LC_MONETARY** category, **LC_NUMERIC** category, and **LC_TIME** category for the locale definition source file format.

LC_MONETARY Category for the Locale Definition Source File Format

Purpose

Defines rules and symbols for formatting monetary numeric information.

Description

The **LC_MONETARY** category of a locale definition source file defines rules and symbols for formatting monetary numeric information. This category begins with an **LC_MONETARY** category header and terminates with an **END LC_MONETARY** category trailer.

All operands for the **LC_MONETARY** category keywords are defined as string or integer values. String values are enclosed by " " (double-quotation marks). All values are separated from the keyword they define by one or more spaces. Two adjacent double-quotation marks indicate an undefined string value. A -1 indicates an undefined integer value. The following keywords are recognized in the **LC_MONETARY** category:

copy
Specifies the name of an existing locale to be used as the definition of this category. If a **copy** statement is included in the file, no other keyword can be specified.

int_curr_symbol
Specifies the string used for the international currency symbol. The operand for the **int_curr_symbol** keyword is a four-character string. The first three characters contain the alphabetic international-currency symbol. The fourth character specifies a character separator between the international currency symbol and a monetary quantity.

currency_symbol
Specifies the string used for the local currency symbol.

mon_decimal_point
Specifies the string used for the decimal delimiter used to format monetary quantities.

mon_thousands_sep
Specifies the character separator used for grouping digits to the left of the decimal delimiter in formatted monetary quantities.

mon_grouping
Specifies a string that defines the size of each group of digits in formatted monetary quantities. The operand for the **mon_grouping** keyword consists of a sequence of semicolon-separated integers. Each integer specifies the number of digits in a group. The initial integer defines the size of the group immediately to the left of the decimal delimiter. The following integers define succeeding groups to the left of the previous group. If the last integer is not -1, the size of the previous group (if any) is repeatedly used for the remainder of the digits. If the last integer is -1, no further grouping is performed.

The following is an example of the interpretation of the **mon_grouping** statement. Assuming the value to be formatted is `123456789` and the operand for the **mon_thousands_sep** keyword is `'` (single-quotation mark), the following results occur:

mon_grouping Value	Formatted Value
`3;-1`	123456’789
`3`	123’456’789
`3;2;-1`	1234’56’789
`3;2`	12’34’56’789
`-1`	123456789

positive_sign
Specifies the string used to indicate a nonnegative-valued formatted monetary quantity.

negative_sign
Specifies the string used to indicate a negative-valued formatted monetary quantity.

int_frac_digits Specifies an integer value representing the number of fractional digits (those after the decimal delimiter) to be displayed in a formatted monetary quantity using the **int_curr_symbol** value.

frac_digits Specifies an integer value representing the number of fractional digits (those after the decimal delimiter) to be displayed in a formatted monetary quantity using the **currency_symbol** value.

p_cs_precedes Specifies an integer value indicating whether the **int_curr_symbol** or **currency_symbol** string precedes or follows the value for a nonnegative formatted monetary quantity. The following integer values are recognized:

- **0** Indicates that the currency symbol follows the monetary quantity.
- **1** Indicates that the currency symbol precedes the monetary quantity.

p_sep_by_space Specifies an integer value indicating whether the **int_curr_symbol** or **currency_symbol** string is separated by a space from a nonnegative formatted monetary quantity. The following integer values are recognized:

- **0** Indicates that no space separates the currency symbol from the monetary quantity.
- **1** Indicates that a space separates the currency symbol from the monetary quantity.
- **2** Indicates that a space separates the currency symbol and the **positive_sign** string, if adjacent.

n_cs_precedes Specifies an integer value indicating whether the **int_curr_symbol** or **currency_symbol** string precedes or follows the value for a negative formatted monetary quantity. The following integer values are recognized:

- **0** Indicates that the currency symbol follows the monetary quantity.
- **1** Indicates that the currency symbol precedes the monetary quantity.

n_sep_by_space Specifies an integer value indicating whether the **int_curr_symbol** or **currency_symbol** string is separated by a space from a negative formatted monetary quantity. The following integer values are recognized:

0 Indicates that no space separates the currency symbol from the monetary quantity.

1 Indicates that a space separates the currency symbol from the monetary quantity.

2 Indicates that a space separates the currency symbol and the **negative_sign** string, if adjacent.

p_sign_posn Specifies an integer value indicating the positioning of the **positive_sign** string for a nonnegative formatted monetary quantity. The following integer values are recognized:

0 Indicates that a **left_parenthesis** and **right_parenthesis** symbol enclose both the monetary quantity and the **int_curr_symbol** or **currency_symbol** string.

1 Indicates that the **positive_sign** string precedes the quantity and the **int_curr_symbol** or **currency_symbol** string.

2 Indicates that the **positive_sign** string follows the quantity and the **int_curr_symbol** or **currency_symbol** string.

3 Indicates that the **positive_sign** string immediately precedes the **int_curr_symbol** or **currency_symbol** string.

4 Indicates that the **positive_sign** string immediately follows the **int_curr_symbol** or **currency_symbol** string.

n_sign_posn Specifies an integer value indicating the positioning of the **negative_sign** string for a negative formatted monetary quantity. The following integer values are recognized:

- **0** Indicates that a **left_parenthesis** and **right_parenthesis** symbol enclose both the monetary quantity and the **int_curr_symbol** or **currency_symbol** string.
- **1** Indicates that the **negative_sign** string precedes the quantity and the **int_curr_symbol** or **currency_symbol** string.
- **2** Indicates that the **negative_sign** string follows the quantity and the **int_curr_symbol** or **currency_symbol** string.
- **3** Indicates that the **negative_sign** string immediately precedes the **int_curr_symbol** or **currency_symbol** string.
- **4** Indicates that the **negative_sign** string immediately follows the **int_curr_symbol** or **currency_symbol** string.

debit_sign Specifies the string used for the debit symbol (**DB**) to indicate a nonnegative formatted monetary quantity.

credit_sign Specifies the string used for the credit symbol (**CR**) to indicate a negative formatted monetary quantity.

left_parenthesis Specifies the character, equivalent to a ((left parenthesis), used by the **p_sign_posn** and **n_sign_posn** statements to enclose a monetary quantity and currency symbol.

right_parenthesis Specifies the character, equivalent to a) (right parenthesis), used by the **p_sign_posn** and **n_sign_posn** statements to enclose a monetary quantity and currency symbol.

A unique customized monetary format can be produced by changing the value of a single statement. For example, the following table shows the results of using all combinations of defined values for the **p_cs_precedes**, **p_sep_by_space**, and **p_sign_posn** statements.

Results of Various Locale Variable Value Combinations				
p_cs_precedes	**p_sign_posn**	**p_sep_by_space =**		
		2	**1**	**0**
p_cs_precedes = 1	**p_sign_posn = 0**	`($1.25)`	`($ 1.25)`	`($1.25)`
	p_sign_posn = 1	`+ $1.25`	`+$ 1.25`	`+$1.25`
	p_sign_posn = 2	`$1.25 +`	`$ 1.25+`	`$1.25+`
	p_sign_posn = 3	`+ $1.25`	`+$ 1.25`	`+$1.25`
	p_sign_posn = 4	`$ +1.25`	`$+ 1.25`	`$+1.25`
p_cs_precedes = 0	**p_sign_posn = 0**	`(1.25 $)`	`(1.25 $)`	`(1.25$)`
	p_sign_posn = 1	`+1.25 $`	`+1.25 $`	`+1.25$`
	p_sign_posn = 2	`1.25$ +`	`1.25 $+`	`1.25$+`
	p_sign_posn = 3	`1.25+ $`	`1.25 +$`	`1.25+$`
	p_sign_posn = 4	`1.25$ +`	`1.25 $+`	`1.25$+`

Example

The following is an example of a possible **LC_MONETARY** category listed in a locale definition source file:

```
LC_MONETARY
#
int_curr_symbol  "<U><S><D>"
currency_symbol  "<dollar-sign>"
mon_decimal_point        "<period>"
mon_thousands_sep        "<comma>"
mon_grouping             <3>
positive_sign            "<plus-sign>"
negative_sign            "<hyphen>"
int_frac_digits  <2>
frac_digits              <2>
p_cs_precedes            <1>
p_sep_by_space   <2>
n_cs_precedes            <1>
n_sep_by_space   <2>
p_sign_posn              <3>
n_sign_posn              <3>
debit_sign               "<D><B>"
credit_sign              "<C><R>"
left_parenthesis         "<left-parenthesis>"
right_parenthesis        "<right-parenthesis>"
#
END LC_MONETARY
```

Implementation Specifics

This category of the locale definition source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **locale** command, **localedef** command.

Character Set Description (charmap) Source File Format , Locale Definition Source File Format , Locale Method Source File Format .

For specific information about other locale categories and their keywords, see the **LC_COLLATE** category, **LC_CTYPE** category, **LC_MESSAGES** category, **LC_NUMERIC** category, and **LC_TIME** category for the locale definition source file format.

LC_NUMERIC Category for the Locale Definition Source File Format

Purpose

Defines rules and symbols for formatting non-monetary numeric information.

Description

The **LC_NUMERIC** category of a locale definition source file defines rules and symbols for formatting non-monetary numeric information. This category begins with an **LC_NUMERIC** category header and terminates with an **END LC_NUMERIC** category trailer.

All operands for the **LC_NUMERIC** category keywords are defined as string or integer values. String values are enclosed by " " (double-quotation marks). All values are separated from the keyword they define by one or more spaces. Two adjacent double-quotation marks indicate an undefined string value. A -1 indicates an undefined integer value. The following keywords are recognized in the **LC_NUMERIC** category:

copy
: The **copy** statement specifies the name of an existing locale to be used as the definition of this category. If a **copy** statement is included in the file, no other keyword can be specified.

decimal_point
: Specifies the string used for the decimal delimiter used to format numeric, non-monetary quantities.

thousands_sep
: Specifies the string separator used for grouping digits to the left of the decimal delimiter in formatted numeric, non-monetary quantities.

grouping
: Defines the size of each group of digits in formatted monetary quantities. The operand for the **grouping** keyword consists of a sequence of semicolon-separated integers. Each integer specifies the number of digits in a group. The initial integer defines the size of the group immediately to the left of the decimal delimiter. The following integers define succeeding groups to the left of the previous group. If the last integer is not -1, the size of the previous group (if any) is used repeatedly for the remainder of the digits. If the last integer is -1, no further grouping is performed.

The following is an example of the interpretation of the **grouping** statement. Assuming the value to be formatted is `123456789` and the operand for the **thousands_sep** keyword is `'` (single quotation mark) the following results occur:

Grouping Value	Formatted Value
`3;-1`	123456’789
`3`	123’456’789
`3;2;-1`	1234’56’789
`3;2`	12’34’56’789
`-1`	123456789

Examples

Following is an example of a possible **LC_NUMERIC** category listed in a locale definition source file:

```
LC_NUMERIC
#
decimal_point     "<period>"
thousands_sep     "<comma>"
grouping          <3>
#
END LC_NUMERIC
```

Implementation Specifics

This category of the locale definition source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **locale** command, **localedef** command.

Character Set Description (charmap) Source File Format , Locale Definition Source File Format , Locale Method Source File Format .

For specific information about other locale categories and their keywords, see the **LC_COLLATE** category, **LC_CTYPE** category, **LC_MESSAGES** category, **LC_MONETARY** category, and **LC_TIME** category for the locale definition source file format.

LC_TIME Category for the Locale Definition Source File Format

Purpose

Defines rules and symbols for formatting time and date information.

Description

The **LC_TIME** category of a locale definition source file defines rules and symbols for formatting time and date information. This category begins with an **LC_TIME** category header and terminates with an **END LC_TIME** category trailer.

Keywords

All operands for the **LC_TIME** category keywords are defined as string or integer values. String values are enclosed by " " (double-quotation marks). All values are separated from the keyword they define by one or more spaces. Two adjacent double-quotation marks indicate an undefined string value. A -1 indicates an undefined integer value. Field descriptors are used by commands and subroutines that query the **LC_TIME** category to represent elements of time and date formats. The following keywords are recognized in the **LC_TIME** category:

copy
: The **copy** statement specifies the name of an existing locale to be used as the definition of this category. If a **copy** statement is included in the file, no other keyword can be specified.

abday
: Defines the abbreviated weekday names corresponding to the `%a` field descriptor. Recognized values consist of 7 semicolon-separated strings. Each string must be of equal length and contain 5 characters or less. The first string corresponds to the abbreviated name (Sun) for the first day of the week (Sunday), the second to the abbreviated name for the second day of the week, and so on.

day
: Defines the full spelling of the weekday names corresponding to the `%A` field descriptor. Recognized values consist of seven semicolon-separated strings. The first string corresponds to the full spelling of the name of the first day of the week (Sunday), the second to the name of the second day of the week, and so on.

abmon
: Defines the abbreviated month names corresponding to the `%b` field descriptor. Recognized values consist of 12 semicolon-separated strings. Each string must be of equal length and contain 5 characters or less. The first string corresponds to the abbreviated name (Jan) for the first month of the year (January), the second to the abbreviated name for the second month of the year, and so on.

mon Defines the full spelling of the month names corresponding to the `%B` field descriptor. Recognized values consist of 12 semicolon-separated strings. The first string corresponds to the full spelling of the name for the first month of the year (January), the second to the full spelling of the name for the second month of the year, and so on.

d_t_fmt Defines the string used for the standard date and time format corresponding to the `%c` field descriptor. The string can contain any combination of characters and field descriptors.

d_fmt Defines the string used for the standard date format corresponding to the `%x` field descriptor. The string can contain any combination of characters and field descriptors.

t_fmt Defines the string used for the standard time format corresponding to the `%X` field descriptor. The string can contain any combination of characters and field descriptors.

am_pm Defines the strings used to represent *ante meridiem* (before noon) and *post meridiem* (after noon) corresponding to the `%p` field descriptor. Recognized values consist of two semicolon-separated strings. The first string corresponds to the *ante meridiem* designation, the last string to the *post meridiem* designation.

t_fmt_ampm Defines the string used for the standard 12-hour time format that includes an **am_pm** value (the `%p` field descriptor). This statement corresponds to the `%r` field descriptor. The string can contain any combination of characters and field descriptors.

era Defines how the years are counted and displayed for each era (or emperor's reign) in a locale, corresponding to the `%E` field descriptor modifier. For each era, there must be one string in the following format:

`direction``:`*`offset`*`:`*`start_date`*`:`*`end_date`*`:`*`name`*`:`*`format`*

The variables for the era-string format are defined as follows:

direction Specifies a **-** (minus sign) or + (plus sign) character. The plus sign character indicates that years count in the positive direction when moving from the start date to the end date. The minus sign character indicates that years count in the negative direction when moving from the start date to the end date.

offset Specifies a number representing the first year of the era.

start_date Specifies the starting date of the era in the *yyyy*/*mm*/*dd* format, where *yyyy*, *mm*, and *dd* are the year, month, and day, respectively. Years prior to the year AD 1 are represented as negative numbers. For example, an era beginning March 5th in the year 100 BC would be represented as `-100/03/05`.

end_date Specifies the ending date of the era in the same form used for the *start_date* variable or one of the two special values **-*** or **+***. A **-*** value indicates that the ending date of the era extends backward to the beginning of time. A **+*** value indicates that the ending date of the era extends forward to the end of time. Therefore, the ending date can be chronologically before or after the starting date of the era. For example, the strings for the Christian eras AD and BC would be entered as follows:

```
+:0:0000/01/01:+*:AD:%o %N
+:1:-0001/12/31:-*:BC:%o %N
```

name Specifies a string representing the name of the era that is substituted for the `%N` field descriptor.

format Specifies a string for formatting the `%E` field descriptor. This string is usually a function of the `%o` and `%N` field descriptors.

An **era** value consists of one string for each era. If more than one era is specified, each era string is separated by a ; (semicolon).

era_year Defines the string used to represent the year in alternate-era format corresponding to the `%Ey` field descriptor. The string can contain any combination of characters and field descriptors.

era_d_fmt Defines the string used to represent the date in alternate-era format corresponding to the `%Ex` field descriptor. The string can contain any combination of characters and field descriptors.

era_t_fmt Defines the alternative time format of the locale, as represented by the `%EX` field descriptor for the **strftime** subroutine.

era_d_t_fmt Defines the alternative date and time format of the locale, as represented by the `%Ec` field descriptor for the **strftime** subroutine.

alt_digits Defines alternate strings for digits corresponding to the `%o` field descriptor. Recognized values consist of a group of semicolon-separated strings. The first string represents the alternate string for 0, the second string represents the alternate string for one, and so on. A maximum of 100 alternate strings can be specified.

Field Descriptors

The **LC_TIME** locale definition source file uses field descriptors to represent elements of time and date formats. Combinations of these field descriptors create other field descriptors or create time-and-date format strings. When used in format strings containing field descriptors and other characters, field descriptors are replaced by their current values. All other characters are copied without change. The following field descriptors are used by commands and subroutines that query the **LC_TIME** category for time formatting:

%a Represents the abbreviated weekday name (for example, Sun) defined by the **abday** statement.

%A Represents the full weekday name (for example, Sunday) defined by the **day** statement.

%b Represents the abbreviated month name (for example, Jan) defined by the **abmon** statement.

%B Represents the full month name (for example, January) defined by the **month** statement.

%c Represents the time-and-date format defined by the **d_t_fmt** statement.

%C Represents the century as a decimal number (00 to 99).

%d Represents the day of the month as a decimal number (01 to 31).

%D Represents the date in %m/%d/%y format (for example, 01/31/91).

%e Represents the day of the month as a decimal number (01 to 31). The `%e` field descriptor uses a two-digit field. If the day of the month is not a two-digit number, the leading digit is filled with a space character.

%Ec Specifies the locale's alternate appropriate date and time representation.

%EC Specifies the name of the base year (period) in the locale's alternate representation.

%Ex Specifies the locale's alternate date representation.

%EX Specifies the locale's alternate time representation.

%Ey Specifies the offset from the `%EC` (year only) field descriptor in the locale's alternate representation.

%EY Specifies the full alternate year representation.

%Od Specifies the day of the month using the locale's alternate numeric symbols.

%Oe Specifies the day of the month using the locale’s alternate numeric symbols.

%OH Specifies the hour (24-hour clock) using the locale’s alternate numeric symbols.

%OI Specifies the hour (12-hour clock) using the locale’s alternate numeric symbols.

%Om Specifies the month using the locale’s alternate numeric symbols.

%OM Specifies the minutes using the locale’s alternate numeric symbols.

%OS Specifies the seconds using the locale’s alternate numeric symbols.

%OU Specifies the week number of the year (Sunday as the first day of the week) using the locale’s alternate numeric symbols.

%Ow Specifies the weekday as a number in the locale’s alternate representation (Sunday = 0).

%OW Specifies the week number of the year (Monday as the first day of the week) using the locale’s alternate numeric symbols.

%Oy Specifies the year (offset from the `%C` field descriptor) in alternate representation.

%h Represents the abbreviated month name (for example, Jan) defined by the **abmon** statement. This field descriptor is a synonym for the `%b` field descriptor.

%H Represents the 24-hour clock hour as a decimal number (00 to 23).

%I Represents the 12-hour clock hour as a decimal number (01 to 12).

%j Represents the day of the year as a decimal number (001 to 366).

%m Represents the month of the year as a decimal number (01 to 12).

%M Represents the minutes of the hour as a decimal number (00 to 59).

%n Specifies a new-line character.

%N Represents the alternate era name.

%o Represents the alternate era year.

%p Represents the a.m. or p.m. string defined by the **am_pm** statement.

%r Represents the 12-hour clock time with a.m./p.m. notation as defined by the **t_fmt_ampm** statement.

%S Represents the seconds of the minute as a decimal number (00 to 59).

%t Specifies a tab character.

%T Represents 24-hour clock time in the format %H:%M:%S (for example, 16:55:15).

%U Represents the week of the year as a decimal number (00 to 53). Sunday, or its equivalent as defined by the **day** statement, is considered the first day of the week for calculating the value of this field descriptor.

%w Represents the day of the week as a decimal number (0 to 6). Sunday, or its equivalent as defined by the **day** statement, is considered as 0 for calculating the value of this field descriptor.

%W Represents the week of the year as a decimal number (00 to 53). Monday, or its equivalent as defined by the **day** statement, is considered the first day of the week for calculating the value of this field descriptor.

%x Represents the date format defined by the **d_fmt** statement.

%X Represents the time format defined by the **t_fmt** statement.

%y Represents the year of the century (00 to 99).

> **Note:** When the environment variable **XPG_TIME_FMT=ON**, **%y** is the year within the century. When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969 to 1999, inclusive); values in the range 00-68 refer to 2000 to 2068, inclusive.

%Y Represents the year as a decimal number (for example, 1989).

%Z Represents the time-zone name, if one can be determined (for example, EST); no characters are displayed if a time zone cannot be determined.

%% Specifies a % (percent sign) character.

Example

The following is an example of a possible **LC_TIME** category listed in a locale definition source file:

```
LC_TIME
#
#Abbreviated weekday names (%a)
abday   "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
        "<T><h><u>";"<F><r><i>";"<S><a><t>"
#
#Full weekday names (%A)
day     "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
        "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
        "<T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
        "<S><a><t><u><r><d><a><y>"
#
#Abbreviated month names (%b)
abmon   "<J><a><n>";"<F><e><b>";"<M><a><r>";"<A><p><r>";\
        "<M><a><y>";"<J><u><n>";"<J><u><l>";"<A><u><g>";\
        "<S><e><p>";"<O><c><t>";"<N><o><v>";"<D><e><c>"
#
#Full month names (%B)
mon     "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
        "<M><a><r><c><h>";"<A><p><r><i><l>";"<M><a><y>";\
        "<J><u><n><e>";"<J><u><l><y>";"<A><u><g><u><s><t>";\
        "<S><e><p><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
        "<N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"
#
#Date and time format (%c)
d_t_fmt "%a %b %d %H:%M:%S %Y"
#
```

```
#Date format (%x)
d_fmt           "%m/%d/%y"
#
#Time format (%X)
t_fmt           "%H:%M:%S"
#
#Equivalent of AM/PM (%p)
am_pm           "<A><M>";"<P><M>"
#
#12-hour time format (%r)
t_fmt_ampm      "%I:%M:%S %p"
#
era             "+:0:0000/01/01:+*:AD:%o %N";\
                "+:1:-0001/12/31:-*:BC:%o %N"
era_year        ""
era_d_fmt       ""
alt_digits      "<0><t><h>";"<1><s><t>";"<2><n><d>";"<3><r><d>";\
                "<4><t><h>";"<5><t><h>";"<6><t><h>";"<7><t><h>";\
                "<8><t><h>";"<9><t><h>";"<1><0><t><h>"
#
END LC_TIME
```

Implementation Specifics

This category of the locale definition source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **locale** command, **localedef** command.

The **strftime** subroutine.

Character Set Description (charmap) Source File Format , Locale Definition Source File Format , Locale Method Source File Format .

For specific information about other locale categories and their keywords, see the **LC_COLLATE** category, **LC_CTYPE** category, **LC_MESSAGES** category, **LC_MONETARY** category, and **LC_NUMERIC** category for the locale definition source file format.

Locale Method Source File Format

Purpose

Specifies the methods to be overridden when constructing a locale.

Description

The **methods** source file maps methods names to the National Language Support (NLS) subroutines that implement those methods. The **methods** file also specifies the libraries where the implementing subroutines are stored.

The methods correspond to those subroutines that require direct access to the data structures representing locale data.

The following is the expected grammar for a **methods** file:

```
method_def : "METHODS"
           | method_assign_list "END METHODS"
           ;

method_assign_list :
        method_assign_list method_assign
        | method_assign_list
        | method_assign
        ;

method_assign :
        "csid" meth_name meth_lib_path
        | "fnmatch" meth_name meth_lib_path
        | "get_wctype" meth_name meth_lib_path
        | "is_wctype" meth_name meth_lib_path
        | "mblen" meth_name meth_lib_path
        | "__mbstopcs" meth_name meth_lib_path
        | "mbstowcs" meth_name meth_lib_path
        | "__mbtopc" meth_name meth_lib_path
        | "mbtowc" meth_name meth_lib_path
        | "__pcstombs" meth_name meth_lib_path
        | "__pctomb" meth_name meth_lib_path
        | "regcomp" meth_name meth_lib_path
        | "regerror" meth_name meth_lib_path
        | "regexec" meth_name meth_lib_path
        | "regfree" meth_name meth_lib_path
        | "rpmatch" meth_name meth_lib_path
        | "strcoll" meth_name meth_lib_path
        | "strfmon" meth_name meth_lib_path
        | "strftime" meth_name meth_lib_path
        | "strptime" meth_name meth_lib_path
        | "strxfrm" meth_name meth_lib_path
        | "towlower" meth_name meth_lib_path
        | "towupper" meth_name meth_lib_path
        | "wcscoll" meth_name meth_lib_path
        | "wcsftime" meth_name meth_lib_path
```

```
        | "wcsid" meth_name meth_lib_path
        | "wcstombs" meth_name meth_lib_path
        | "wcswidth" meth_name meth_lib_path
        | "wcsxfrm" meth_name meth_lib_path
        | "wctomb" meth_name meth_lib_path
        | "wcwidth" meth_name meth_lib_path
        ;

meth_name: global_name
         | cfunc_name
         ;

global_name: "CSID_STD"
        | "FNMATCH_C"
        | "FNMATCH_STD"
        | "GET_WCTYPE_STD"
        | "IS_WCTYPE_SB"
        | "IS_WCTYPE_STD"
        | "LOCALECONV_STD"
        | "MBLEN_932"
        | "MBLEN_EUCJP"
        | "MBLEN_SB"
        | "__MBSTOPCS_932"
        | "__MBSTOPCS_EUCJP"
        | "__MBSTOPCS_SB"
        | "MBSTOWCS_932"
        | "MBSTOWCS_EUCJP"
        | "MBSTOWCS_SB"
        | "__MBTOPC_932"
        | "__MBTOPC_EUCJP"
        | "__MBTOPC_SB"
        | "MBTOWC_932"
        | "MBTOWC_EUCJP"
        | "MBTOWC_SB"
        | "NL_MONINFO"
        | "NL_NUMINFO"
        | "NL_RESPINFO"
        | "NL_TIMINFO"
        | "__PCSTOMBS_932"
        | "__PCSTOMBS_EUCJP"
        | "__PCSTOMBS_SB"
        | "__PCTOMB_932"
        | "__PCTOMB_EUCJP"
        | "__PCTOMB_SB"
        | "REGCOMP_STD"
        | "REGERROR_STD"
        | "REGEXEC_STD"
        | "REGFREE_STD"
        | "RPMATCH_C"
        | "RPMATCH_STD"
        | "STRCOLL_C"
        | "STRCOLL_SB"
        | "STRCOLL_STD"
        | "STRFMON_STD"
        | "STRFTIME_STD"
        | "STRPTIME_STD"
        | "STRXFRM_C"
        | "STRXFRM_SB"
        | "STRXFRM_STD"
        | "TOWLOWER_STD"
        | "TOWUPPER_STD"
        | "WCSCOLL_C"
```

```
| "WCSCOLL_STD"
| "WCSFTIME_STD"
| "WCSID_STD"
| "WCSTOMBS_932"
| "WCSTOMBS_EUCJP"
| "WCSTOMBS_SB"
| "WCSWIDTH_932"
| "WCSWIDTH_EUCJP"
| "WCSWIDTH_LATIN"
| "WCSXFRM_C"
| "WCSXFRM_STD"
| "WCTOMB_932"
| "WCTOMB_EUCJP"
| "WCTOMB_SB"
| "WCWIDTH_932"
| "WCWIDTH_EUCJP"
| "WCWIDTH_LATIN"
;
```

Where **cfunc_name** is the name of a user supplied subroutine, and **meth_lib_path** is an optional path name for the library containing the specified subroutine.

The **localedef** command parses this information to determine the methods to be used for this locale. The following subroutines must be specified in the **method** file:

- **__mbtopc**
- **__mbstopcs**
- **__pctomb**
- **__pcstombs**
- **mblen**
- **mbstowcs**
- **mbtowc**
- **wcstombs**
- **wcswidth**
- **wctomb**
- **wcwidth**

Any other method not specified in the **method** file retains the default.

Mixing of **cfunc_name** values and **global_name** values is not allowed. A **method** file should not include both. If the **localedef** command receives a **method** file containing both **cfunc_name** values and **global_name** values, an error is generated and the locale is not created.

It is not mandatory that the **METHODS** section specify the library name. If an individual method does not specify a library, the method inherits the most recently specified library. The **libc.a** library is the default library.

The method for the **mbtowc** and **wcwidth** subroutines should avoid calling other methods where possible.

An understanding of how the **__mbtopc**, **__mbstopcs**, **__pctomb**, and **__pcstombs** subroutines process wide characters is useful when constructing a **method** file. These subroutines should not be used in applications programs.

__mbtopc Subroutine

The **__mbtopc** subroutine converts a character to a process code.

The syntax for the **__mbtopc** subroutine is as follows:

size_t __mbtopc(*PC***,** *S***,** *LenS***,** *Err***)**
wchar_t ***PC*;**
uchar ***S*;**
size_t *LenS***;**
int ***Err*;**

The input buffer pointed to by the *S* parameter contains the number of bytes of character data specified in the *LenS* parameter. The **__mbtopc** subroutine attempts to convert the character to a process code. If a valid character is found in the input buffer pointed to by the *S* parameter, the character is converted and stored in the *PC* parameter, and the number of bytes in the character is returned.

If the number of bytes specified by the *LenS* parameter in the input buffer pointed to by the *S* parameter form an invalid character, the subroutine returns 0 and sets the *Err* parameter to the value -1. If a character cannot be formed in the number of bytes specified by the *LenS* parameter or less, the subroutine returns 0 and sets the *Err* parameter to the number of bytes required to form a character beginning with the data pointed to by the *S* parameter.

The parameters have the following values:

PC	Points to a wide character to contain the converted character.
S	Points to the buffer of character data to be converted.
LenS	Specifies the number of bytes of character data pointed to by the *S* parameter.
Err	Specifies an error value indicating why the conversion failed.

__mbstopcs Subroutine

The **__mbstopcs** subroutine converts a character string to a process code string.

The syntax for the **__mbstopcs** subroutine is as follows:

size_t __mbstopcs(*PC***,** *LenPC***,** *S***,** *LenS***,** *StopCh***,** *EndPtr***,** *Err***)**
wchar_t ***PC*;**
size_t *LenPC***;**
uchar ***S*;**
size_t *LenS***;**
uchar *StopCh***;**
uchar ****EndPtr*;**
int ***Err*;**

The input buffer pointed to by the *S* parameter contains the number of bytes of character data specified in the *LenS* parameter. The **__mbstopcs** subroutine attempts to convert the character data to process codes. The conversion of characters continues until one of the following occurs:

- The number of bytes specified by the *LenS* parameter have been converted.
- The number of characters specified by the *LenPC* parameter have been converted.
- The byte value specified in the *StopCh* parameter is encountered in the input buffer pointed to by the *S* parameter.
- An invalid or incomplete character is found in the input buffer pointed to by the *S* parameter.

If the number of bytes specified by the *LenS* parameter or the number of characters specified by the *LenPC* parameter are successfully converted, the **__mbstopcs** subroutine returns the number of characters converted, sets the *Err* parameter to 0, and sets the *EndPtr* parameter to point immediately after the last character converted in the input buffer pointed to by the *S* parameter.

If the byte specified by the *StopCh* parameter is found in the input buffer pointed to by the *S* parameter, the following occurs:

- Conversion ceases.
- The value specified by the *StopCh* parameter is placed in the *PC* parameter.
- The *EndPtr* parameter is set to point immediately after the value specified by the *StopCh* parameter.
- The *Err* parameter is set to 0.
- The number of characters converted is returned.

If an invalid character is found in the input buffer pointed to by the *S* parameter, the *EndPtr* parameter is set to point to the start of this character, the *Err* parameter is set to (**size_t**)-1, and the **__mbstopcs** subroutine returns the number of characters converted.

If an incomplete character is found at the end of the input buffer pointed to by the *S* parameter, the *EndPtr* parameter is set to point to the start of the incomplete character, and the *Err* parameter is set to the number of bytes in a character starting with the byte pointed to by *EndPtr* parameter. The **__mbstopcs** subroutine returns the number of characters converted.

The parameters have the following values:

PC	Points to a **wchar_t** array to contain the converted characters.
LenPC	Specifies the maximum number of wide characters that can be placed in the *PC* parameter.
S	Points to a buffer of character data to be converted.
LenS	Specifies the number of bytes of character data in the *S* parameter.
StopCh	Specifies a single-byte character value to indicate end of data in the *S* parameter.
EndPtr	Points into the *S* parameter where character conversion ended.
Err	Specifies an error value indicating why the conversion failed.

__pctomb Subroutine

The **__pctomb** subroutine converts a process code to a character.

The syntax for the **__pctomb** subroutine is as follows:

size_t __pctomb(*S***,** *LenS***,** *PC***,** *Err***)**
char ****S*;**
size_t *LenS***;**
wchar_t ****PC*;**
int ****Err*;**

The input buffer pointed to by the *PC* parameter contains a wide character that the subroutine attempts to convert to a character in the input buffer pointed to by the *S* parameter. If a valid process code is found in the input buffer pointed to by the *PC* parameter, it is converted and stored in the input buffer pointed to by the *S* parameter, and the number of bytes in the character is returned.

If the wide character in the input buffer pointed to by the *PC* parameter is invalid, the **__pctomb** subroutine returns 0 and sets the *Err* parameter to the value (**size_t**)-1. If the length of the character is greater than the number of bytes specified by the *LenS* parameter, the **__pctomb** subroutine returns 0 and sets the *Err* parameter to the number of bytes required to form the character.

The parameters have the following values:

S	Points to a buffer to contain the converted process code.
LenS	Specifies the size of the character array pointed to by the *S* parameter.
PC	Points to the wide character to be converted.
Err	Specifies an error value indicating why the conversion failed.

__pcstombs Subroutine

The **__pcstombs** subroutine converts a wide character string to a character string.

The syntax for the **__pcstombs** subroutine is as follows:

size_t __pcstombs(*S***,** *LenS***,** *PC***,** *LenPC***,** *StopCh***,** *EndPtr***,** *Err***)**
char ****S*;**
size_t *LenS***;**
wchar_t ****PC*;**
size_t *LenPC***;**
wchar_t *StopCh***;**
char *****EndPtr*;**
int ****Err*;**

The input buffer pointed to by the *PC* parameter contains the number of wide characters specified by the *LenPC* parameter. The **__pcstombs** subroutine attempts to convert the process codes to characters. The conversion continues until one of the following occurs:

- The number of wide characters specified by the *LenPC* parameter have been converted.
- The number of bytes specified by the *LenS* parameter have been converted.
- The character value specified in the *StopCh* parameter is encountered in the input buffer pointed to by the *PC* parameter.
- An invalid wide character is found in the input buffer pointed to by the *PC* parameter.

If the number of bytes specified by the *LenS* parameter or the number of characters specified by the *LenPC* parameter are successfully converted, the **__pcstombs** subroutine returns the number of bytes placed in the buffer pointed to by the *S* parameter, sets the *Err* parameter to 0, and sets the *EndPtr* parameter to point immediately after the last character converted in the input buffer pointed to by the *PC* parameter.

If the character specified by the *StopCh* parameter is found in the input buffer pointed to by the *PC* parameter, the following occurs:

- Conversion ceases.
- The character specified by the *StopCh* parameter is placed at the end of the data currently pointed to by the *S* parameter.
- The *EndPtr* parameter is set to point immediately after the character specified by the *StopCh* parameter.
- The *Err* parameter is set to 0.
- The number of bytes placed in the buffer pointed to by the *S* parameter is returned.

If an invalid wide character is found in the input buffer pointed to by the *PC* parameter, the *EndPtr* parameter is set to point to the start of this character, the *Err* parameter is set to (**size_t**)-1, and the **__pcstombs** subroutine returns the number of bytes placed in the buffer pointed to by the *S* parameter.

The parameters have the following values:

S	Points to a buffer to contain the converted data.
LenS	Specifies the size in bytes of the character array pointed to by the *S* parameter.
PC	Points to a **wchar_t** array to be converted.
LenPC	Specifies the number of wide characters in the array pointed to by the *PC* parameter.
StopCh	Specifies a wide-character value to indicate end of data in the array pointed to by the *PC* parameter.
EndPtr	Points into the *S* parameter where character conversion ended.
Err	Specifies the error value indicating why the conversion failed.

Implementation Specifics

This source file format is part of the Base Operating System (BOS) Runtime.

Files

/usr/lib/nls/loc/*	Specifies locale definition source files for supported locales.
/usr/lib/nls/charmap/*	Specifies character set description (**charmap**) source files for supported locales.

Related Information

The **locale** command, **localedef** command.

Character Set Description (charmap) Source File Format , Locale Definition Source File Format .

For specific information about other locale categories and their keywords, see the **LC_COLLATE** category, **LC_CTYPE** category, **LC_MESSAGES** category, **LC_MONETARY** category, **LC_NUMERIC**, and **LC_TIME** category for the locale definition source file format.

magic File Format

Purpose

Defines file types.

Description

The **/etc/magic** file is used by commands such as the following to determine the type of a given file:

- **file** command
- **more** command

Entering the following command would result in a printed message describing the file type of the *FileName* parameter*:*

```
file FileName
```

If *FileName* contains a byte pattern corresponding to an executable file, the pattern would match a stanza in the **/etc/magic** file and the `executable` message would be displayed. If the *FileName* is a data file, a `data` message is displayed, and so on.

The fields of the magic file are as follows:

1. Byte offset
2. Value type
3. Optional relational operator ("=" by default) and value to match (numeric or string constant)
4. String to be printed

Numeric values may be decimal, octal, or hexadecimal. Strings can be entered as hexadecimal values by preceding them with ’0x’.

The last string can have one **printf** format specification.

The > (greater than) symbol in occasional column 1s is magic; it forces commands to continue scanning and matching additional lines. The first line not marked with the > sign terminates the search.

Examples

```
0 short 2345 this is a dummy type file
```

```
0 long 0x1234 this is a different dummy type file
```

```
>12 long >0 another possible type
```

```
0 short 7895 last type of file
```

Related Information

.mailrc File Format

Purpose

Sets defaults for the **mail** command.

Description

The **.mailrc** file can be placed in your **$HOME** directory to personalize the **Mail** program. You can create the **.mailrc** file with any ASCII editor. Once the file is created, the **Mail** program reads the file when you send or read mail, and applies the options you have set. In the file, you can define aliases for other users' mail addresses. You can also change the way mail is displayed and stored on your system.

The **Mail** program uses a master file in the same format, **/usr/share/lib/Mail.rc**. Options you set in your **$HOME/.mailrc** file override comparable options in the **Mail.rc** file.

A line that begins with a # (pound sign) followed by a space is treated as a comment. The Mail program ignores the entire line and any entries or options it contains.

Entries

Use the following **mail** subcommands as entries in the **.mailrc** file:

alias *NewAlias* { *Address...* | *PreviousAlias...* }

Defines an alias or distribution list. The alias can be defined as an actual mail address, or as another alias defined in a previous entry in the **.mailrc** file. To define a group, enter multiple addresses or previous aliases separated by spaces.

ignore *FieldList*
Adds the header fields in the *FieldList* parameter to the list of fields to be ignored. Ignored fields are not displayed when you look at a message with the **type** or **print** subcommand. Use this subcommand to suppress machine-generated header fields. Use the **Type** or **Print** subcommand to print a message in its entirety, including ignored fields.

set [*OptionList* | *Option*=*Value*...]

Sets an option. The argument following the **set** option can be either an *OptionList* giving the name of a binary option (an option that is either set or unset) or an *Option*=*Value* entry used to assign a value to an option.

unset *OptionList*
Disables the values of the options specified in *OptionList*. This action is the inverse of the **set** *OptionList* entry.

Binary Options for the set and unset Entries

Use the **set** entry to enable options and the **unset** entry to disable options. Add the options you want to set or unset to the **$HOME/.mailrc** file. The options and the actions they generate are as follows:

append Adds messages saved in your mailbox to the end rather than to the beginning of the **$HOME/mbox** file.

ask Prompts for the subject of each message sent. If you do not wish to create a subject field, press the Enter key at the prompt.

askcc Prompts for the addresses of people who should receive copies of the message. If you do not wish to send copies, press the Enter key at the prompt.

autoprint Sets the **delete** subcommand to delete the current message and display the next message.

debug Displays debugging information. Messages are not sent while in debug mode. This is the same as specifying the **-d** flag on the command line.

dot Interprets a period entered on a line by itself as the end of a message you are sending.

hold Holds messages that you have read but have not deleted or saved in the system mailbox instead of in your personal mailbox. This option has no effect on deleted messages.

ignore Ignores interrupt messages from your terminal and echoes them as @ (at sign) characters.

ignoreeof Sets the **mail** command to refuse the Ctrl-D key sequence as the end of a message.

keepsave Prevents the Mail program from deleting messages that you have saved with the **s** or **w** mailbox subcommand. Normally, messages are deleted automatically when you exit the **mail** command. Use the **keepsave** and **hold** options to hold messages in your system mailbox. Otherwise, the messages are placed in your personal mailbox (**$HOME/mbox**).

metoo Includes the sender in the alias expansion. By default, expanding the alias removes the sender. When this option is set in your **.mailrc** file, sending a message using an alias that includes your name sends a copy of the message to your mailbox.

noheader Suppresses the list of messages in your mailbox when you start the Mail program. Instead, only the mailbox prompt (&) is displayed. To get a list of messages, use the **h** mailbox subcommand.

nosave Prevents retention of interrupted letters in the **$HOME/dead.letter** file.

quiet Suppresses the printing of the banner when the **Mail** program starts. The banner is the line that shows the name of the **Mail** program.

Replyall Reverses the meaning of the **reply** subcommand and the **Reply** subcommand.

verbose Displays the actual delivery of messages on the terminal. This is the same as specifying the **-v** flag on the command line.

Value Options for the set Entry

You can use a **set** entry to assign values to the following options. For example, enter `set screen=20` to limit headers to 20 lines per screen.

crt=*Lines*	Defines the number of lines of a mail message the Mail program displays before pausing for input (this option starts the **pg** command to control the scrolling).
EDITOR=*Editor*	Gives the full path name of the editor to be started with the **e** mailbox subcommand or the **~e** mail editor subcommand. The default editor is **/usr/bin/e**.
escape=*Character*	Changes the escape character used for mail editor subcommands. The default character is ~ (tilde).
folder=*PathName*	Gives the path name of a directory in which to store mail folders. Once the directory is defined, you can use the + (plus sign) notation to refer to it when using the *FileName* parameter with mailbox subcommands.
record=*FileName*	Defines a file in which to record outgoing mail. The path name must be absolute (that is, a full path name), or be given relative to the current directory. **Note:** If you set up a file to record outgoing messages, read the file periodically with the **mail -f** command and delete unnecessary messages. Otherwise, the file will grow and eventually use all of your storage space.
screen=*Lines*	Defines the number of lines of message headers displayed (for example, in response to the **h** mailbox subcommand) before pausing for input.
toplines=*Lines*	Defines the number of lines displayed by the **top** mailbox subcommand.
VISUAL=*Editor*	Gives the full path name of the editor to be started with the **v** mailbox subcommand or the **~v** mail editor subcommand. The default editor is **/usr/bin/vi**.

Examples

1. To ignore the `Message-ID` field and the `Received` field, place the following entry in the **.mailrc** file:

   ```
   ignore message-id received
   ```

   When messages are displayed in the mailbox, the machine message ID number and the date your system received the message are not displayed.

2. To set a folder directory, place the following entry in the **.mailrc** file:

   ```
   set folder=/home/kaye/notes
   ```

   To save message `1` from the mailbox in the folder `procedures`, enter the following at the mailbox prompt (&):

```
s 1 +procedure
```

Message `1` is saved in the `/home/kaye/notes/procedures` file (if the file already exists, the message is appended to the file).

3. To record outgoing mail in a folder directory, place the following pair of entries in the **.mailrc** file:

   ```
   set record=/home/pierre/letters/mailout
   set folder=/home/pierre/letters
   ```

   Outgoing mail is placed in the `/home/pierre/letters/mailout` file, and can be read with the following command:

   ```
   mail -f +mailout
   ```

4. To combine the delete and print commands and also instruct the Mail program to include your user ID when expanding aliases, enter the following in your **.mailrc** file:

   ```
   set autoprint metoo
   ```

   The **autoprint** option causes the next message to be displayed whenever you delete a message. The **metoo** option causes the Mail program to send a copy of messages to you when it expands mail aliases. By default, the Mail program discards your user address when it expands an alias, so that you do not get a copy of mail you send.

5. To unset an option that is set in the **/usr/share/lib/Mail.rc** file, enter the following in your **.mailrc** file:

   ```
   unset askcc
   ```

   This entry prevents the mail editor from requesting a carbon copy list when you create messages, even if the **askcc** option is set in the **Mail.rc** file.

6. To set aliases for two users and a distribution list that includes several users, enter the following in your **.mailrc** file:

   ```
   alias george george@thor.valhalla.dbm.comm
   alias bill @odin.UUCP:@depta.UCCP:@deptb:bill@deptc
   alias mygroup amy@cleo george bill
   ```

   To send mail to user `bill` using his alias, enter:

   ```
   mail bill
   ```

   To send mail to everyone in the `mygroup` list, enter:

   ```
   mail mygroup
   ```

   When you complete and send the message, the **mail** command actually addresses it as follows:

   ```
   amy@cleo george@thor.valhalla.dbm.comm @odin.UUCP:@depta.UCCP:
   @deptb:bill@deptc
   ```

Implementation Specifics

This file is part of the Base Operating System.

Files

/usr/share/lib/Mail.rc	Contains systemwide defaults for the Mail program.
$HOME/.mailrc	Contains user-specific defaults for the Mail program.

Related Information

The **mail** command, **pg** command.

Mail Editor Subcommands for the mail, Mail Command.

Mailbox Subcommands for the mail, Mail Command.

map3270 File Format for TCP/IP

Purpose

Defines keyboard mapping and colors for the **tn3270** command.

Description

The **/etc/map3270** file defines keyboard mapping and colors for the **tn3270** command. When emulating 3270 terminals, mapping must be performed between key sequences entered on a user's (ASCII) keyboard and the keys that are available on a 3270 emulator.

For example, the 3270 emulator key **EEOF** erases the contents of the current field from the location of the cursor to the end of the field. In order to accomplish this function, the terminal user and a program emulating a 3270 emulator must be compatible with regard to what keys invoke the **EEOF** function.

The requirements for these sequences are:

- The first character of the sequence is outside of the standard ASCII printable characters.
- No one sequence is an initial part of another (although sequences may share initial parts).

The **/etc/map3270** file consists of entries for various terminals. The first part of an entry lists names of terminals using that entry. These names should be the same as those in the **/usr/share/lib/terminfo/*.ti** files.

Note: Often, several terminals from different **/usr/share/lib/terminfo/*.ti** entries use the same **/etc/map3270** file entry. For example, both `925` and `925vb` (for 925 with visual bells) might use the same **map3270** file entry. Each name is separated by a `|` (vertical bar), after which comes a `{` (left brace); the definitions; and finally, a `}` (right brace).

Format

The definitions begin with a reserved keyword, which identifies the 3270 function. The keyword is followed by an = (equal sign), which in turn is followed by the various string sequences to generate the particular function. The definitions end with a ; (semi-colon). The string sequences are printable ASCII characters enclosed inside ' ' (single quotes) and separated by | (vertical bars).

Special characters can be used within ' ' (single quotes). A ^ (caret) indicates a control character. For example, the string `'^a'` represents Ctrl-A; that is, hexadecimal 1 (the string `'^A'` generates the same code). To generate `delete` or `rubout`, enter `'^d' '^?'` (Ctrl-D or Ctrl-?). To represent a control character in the **/etc/map3270** file, you must use the caret. Typing `Control-A` or `Ctrl-A` does not work.

Note: The Ctrl-^ key sequence (to generate a hexadecimal 1E) is represented as `'^^'` (not `'^\^'`).

The \ (backslash) special character precedes other characters to change their meaning. Because this has little effect for most characters, its use is not recommended. The backslash prevents a single quote from terminating a string, for example the string `'^\''` represents Ctrl-'. For a backslash to be part of a string, place two backslashes (`'\\'`) in the string.

In addition, the following characters are special:

```
'\e'   Specifies an escape character.
'\n'   Specifies a new line.
'\t'   Specifies a tab.
'\r'   Specifies a carriage return.
```

It is not necessary for each character in a string to be enclosed within single quotes. The string `'\e\e\e'` means three escape characters.

Comments, which may appear anywhere on a line, begin with a # (pound sign) and terminate at the end of that line. However, comments cannot begin inside a quoted string. A pound sign inside a quoted string has no special meaning.

3270 Keys Supported

Note: Some of the following keys do not exist on a 3270 emulator. The functions listed with an * (asterisk) are not supported by the **tn3270** command. An unsupported function causes the **tn3270** command to send a bell sequence to the user's terminal.

The **/etc/map3270** file supports the following list of 3270 key names:

Key Name	Functional Description
altk*	Alternate keyboard dvorak
aplend*	Treat input as ASCII
aploff*	APL off
aplon*	APL on
attention	Attention key. The attention key sends an IAC BREAK TELNET protocol sequence to the TELNET server on a VM or MVS system. The TELNET server is responsible for implementing the attention key.
btab	Field tab back
clear	Local clear of the 3270 screen
clrtab	Clear all column tabs
colbak	Column back tab
coltab	Column tab
cursel*	Cursor select
delete	Delete character
deltab	Delete a column tab

disc	Disconnect (suspend)
down	Down cursor
dp	Duplicate character
eeof	Erase end of field
einp	Erase input
enter	Enter key
erase	Erase last character
escape	Enter TELNET command mode
ferase	Erase field
fieldend	Tab to last non-blank of current or next unprotected (writable) field
flinp	Flush input
fm	Field mark character
home	Home the cursor
indent	Indent one tab stop
init*	New terminal type
insrt	Toggle insert mode
left	Left cursor
lprt*	Local print
master_reset	Reset, unlock, and redisplay
nl	New line
pa1	Program attention 1
pa2	Program attention 2
pa3	Program attention 3
pfk1	Program function key 1
pfk2	Program function key 2
.	.
.	.
.	.
pfk36	Program function key 36.
pcoff*	Xon/xoff off

pcon*	Xon/xoff on
reset	Reset key-unlock keyboard
reshow	Redisplay the screen
right	Right cursor
sethom	Set home position
setmrg	Set left margin
settab	Set a column tab
synch	In synch with the user
tab	Field tab
treq	Test request
undent	Undent one tab stop
up	Up cursor
werase	Erase last word
wordbacktab	Tab to beginning of current or last word
wordend	Tab to end of current or next word
wordtab	Tab to beginning of next word
xoff*	Hold output
xon*	Release output

A Sample Entry

The following default entry is included within the **tn3270** command and is used when it is unable to locate a version in the user’s environment or the **/etc/map3270** file.

```
name {                      # actual name comes from TERM variable
clear = ’^z’;
flinp = ’^x’;
enter = ’^m’;
delete = ’^d’ | ’^?’;       # note that ’^?’ is delete (rubout)
synch = ’^r’;
reshow = ’^v’;
eeof = ’^e’;
tab = ’^i’;
btab = ’^b’;

nl = ’^n’;
left = ’^h’;
right = ’^l’;
up = ’^k’;
down = ’^j’;
einp = ’^w’;
reset = ’^t’;
```

```
xoff = ’^s’;
xon = ’^q’;
escape = ’^c’;
ferase = ’^u’;
insrt = ’ ’;
# program attention keys
pa1 = ’^p1’; pa2 = ’^p2’; pa3 = ’^p3’;
# program function keys
pfk1 = ’1’; pfk2 = ’2’; pfk3 = ’3’; pfk4 = ’4’;
pfk5 = ’5’; pfk6 = ’6’; pfk7 = ’7’; pfk8 = ’8’;
pfk9 = ’9’; pfk10 = ’  ’; pfk11 = ’-’; pfk12 = ’=’;
pfk13 = ’’; pfk14 = ’@’; pfk15 = ’0;
pfk17 = ’’; pfk18 = ’’; pfk19 = ’’; pfk20 = ’;
pfk21 = ’ pfk22 = ’)’; pfk23 = ’_’; pfk24 = ’ ’;
}
```

3270 Key Definitions

The following table shows the proper keys to emulate each 3270 function when using the default key mapping supplied with the **tn3270** command.

3270 Key Definitions		
Function	**3270 Key**	**Default Key(s)**
Command Keys	Enter	RETURN
	Clear	Ctrl-z
	Attention	Ctrl-F12
Cursor Movement Keys	New line	Ctrl-n or Home
	Tab	Ctrl-i
	Back tab	Ctrl-b
	Cursor left	Ctrl-h
	Cursor right	Ctrl-l
	Cursor up	Ctrl-k
	Cursor down	Ctrl-j or LINE FEED
Edit Control Keys	Delete char	Ctrl-d or RUB
	Erase EOF	Ctrl-e
	Erase input	Ctrl-w
	Insert mode	ESC Space
	End insert	ESC Space
Program Function Keys	PF1	ESC 1
	PF2	ESC 2

	...	...
	PF10	ESC 0
	PF11	ESC -
	PF12	ESC =
	PF13	ESC !
	PF14	ESC @
	...	...
	PF24	ESC +
Program Attention Keys	PA1	Ctrl-p 1
	PA2	Ctrl-p 2
	PA3	Ctrl-p 3
Local Control Keys	Reset after error	Ctrl-r
	Purge input buffer	Ctrl-x
	Keyboard unlock	Ctrl-t
	Redisplay screen	Ctrl-v
Other Keys	Erase current field	Ctrl-u

Files

/etc/3270.keys	Contains the default keyboard mapping.
/usr/share/lib/terminfo/*.ti	Files containing terminal information.

Related Information

The **telnet**, **tn**, or **tn3270** command.

The **.3270keys** file format.

Maxuuscheds File Format for BNU

Purpose

Limits the number of instances of the **uusched** and **uucico** daemons that can run simultaneously.

Description

The **/etc/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 for 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/locks directory	Contains lock files that prevent multiple uses of devices and multiple calls to systems.
/etc/uucp directory	Contains some of the configuration files for BNU, including the **Maxuuscheds** file.

Related Information

The **uucico** daemon.

Maxuuxqts File Format for BNU

Purpose

Limits the number of instances of the BNU **uuxqt** daemon that can run simultaneously on the local system.

Description

The **/etc/uucp/Maxuuxqts** file limits both the number of instances of the Basic Networking Utilities (BNU) **uuxqt** daemon that can run simultaneously on the local system and the number of commands from remote systems that can run at one time.

This file contains an ASCII number that can be changed for your installation. The default value is 2. 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp directory	Contains some of the configuration files for BNU, including the **Maxuuxqts** file.

Related Information

The **uuxqt** daemon.

.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 (customarily the **$HOME/.mh_alias** file) and a systemwide alias file, the **/etc/mh/MailAliases** file. Depending on the MH configuration, aliases may also be defined in the **/etc/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 **/etc/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. Otherwise, you must use the **-alias** flag each time you use an MH command that requires 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	Specifies a simple address.
Address	Specifies a simple Internet-style address.
Group	Specifies a group name (or number) from the **/etc/group** file.
Alias-File	Specifies a system 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	List of addresses that make up a group.
<Alias-File	System file to be read for more alias definitions.

The addresses in the *AddressList* variable must be separated by commas.

Note: If there are references to aliases within an alias definition, those aliases must be defined in a following line of the alias file.

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 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 ID of the group. Each login name appearing in the **/etc/passwd** file that matches the address group is added to the address list for the alias.
* (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 **/etc/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. The system 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.

Examples

The following example of an **.mh_alias** file illustrates some of its features:

```
</home/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 `/home/sarah/morealiases` file. The `systems` alias is defined as all users listed as members of the group `systems` in the **/etc/group** file. The `staff` alias is defined as all users whose group ID in the **/etc/passwd** file is equivalent to the `staff` group. Finally, the `everyone` alias is defined as all users with a user ID in the **/etc/passwd** file greater than 200.

The `manager` alias is defined as an alias for user `harold@harold`. The `project` alias is defined as the users `lance`, `mark@remote`, `peter`, and `manager`.

Implementation Specifics

This file format is part of Message Handler in the Base Operating System.

Files

/etc/aliases	Contains systemwide aliases for the **sendmail** command.
/etc/group	Contains basic group attributes.
/etc/passwd	Contains user authentication information.
/etc/mh/MailAliases	Contains the defaults alias file for systemwide aliases, which is set up by a user with root user authority.
/etc/mh/mtstailor	Tailors the Message Handler (MH) environment to the local environment.
.mh_profile	Customizes the Message Handler (MH) package.

Related Information

The **aliases** file, **/etc/group** file, **/etc/passwd** file, **$HOME/.mh_profile** file.

mib.defs File Format

Purpose

Provides descriptions of Management Information Base (MIB) variables for the **snmpinfo** command.

Description

The **mib.defs** file provides object descriptions of MIB variables for the **snmpinfo** command issued with the **get**, **next**, **set,** and **dump** options. See the **snmpinfo** command for more information.

The **mib.defs** file is not intended to be edited by the user. The file should be created with the **mosy** command. See the **mosy** command for information on how to create the **mib.defs** file. This file has the following format:

The MIB group fields are separated by spaces or tabs and contain the following information:

GroupDescriptor	Holds the textual name of the MIB group.
GroupEntry	Denotes the parent MIB group and the location of this MIB group in the parent group. This field is used by the **snmpinfo** command to resolve the ASN.1 dotted notation for MIB variables under this group.

The MIB groups are defined as follows:

Group Descriptor	Group Entry
internet	**iso.3.6.1**
directory	**internet.1**
mgmt	**internet.2**
.	.
.	.
.	.
mib-2	**mgmt.1**
system	**mib-2.1**
.	.
.	.

The object definitions of MIB variables are formatted as follows:

Object Descriptor	Group Entry	Syntax	Access	Status
sysDescr	**system.1**	**DisplayString**	**read-only**	**mandatory**

The MIB variable fields are separated by spaces or tabs, and contain the following information:

ObjectDescriptor Holds the textual name of the object.

GroupEntry Denotes the MIB object group and the location of this MIB variable in this group. This field is used by the **snmpinfo** command to resolve the ASN.1 dotted notation for this MIB variable.

Syntax Denotes the type of the object as one of the following:

- INTEGER
- OCTET STRING or DisplayString
- OBJECT IDENTIFIER
- Network Address
- Counter
- Gauge
- TimeTicks
- Opaque

Access Designates the access permissions for the object and can be one of the following:

- Read-only
- Read-write
- Write-only
- Not-accessible

Status Designates the RFC 1213 compliance status of the object and can be one of the following:

- Mandatory
- Optional
- Deprecated
- Obsolete

The parent MIB group definition required for a particular MIB variable *GroupEntry* definition must precede the object definition for the MIB variable.

Comments begin with a # (pound sign) or **- -** (two dashes) and continue to the end of the line.

Implementation Specifics

This command is part of Simple Network Management Protocol Agent Applications in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/usr/samples/snmpd/smi.my	Defines the ASN.1 definitions by which the SMI is defined as in RFC 1155.
/usr/samples/snmpd/mibII.my	Defines the ASN.1 definitions for the MIB II variables as defined in RFC 1213.

Related Information

The **mosy** command, **snmpinfo** command.

named.conf File Format for TCP/IP

Purpose

Defines the configuration and behavior of the **named** daemon.

Description

The **/etc/named.conf** file is the default configuration 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.

> **Note:** The **named** daemon reads the configuration file only when the **named** daemon starts or when the **named** daemon receives an SRC **refresh** command or a **SIGHUP** signal.

The data in the **named.conf** file specifies general configuration characteristics for the name server, defines each zone for which the name server is responsible (its zones of authority), and provides further config information per zone, possibly including the source DOMAIN database file for the zone.

Any database files referenced in the **named.conf** file must be in Standard Resource Record Format. These data files can have any name and any directory path. 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.

Format

General

Comments in the **named.conf** file can begin with a # (pound sign) or // (two forward slashes), or can be enclosed in the C-style comment characters, e.g., /* comment text */.

Configuration options are lines of text beginning with a keyword, possibly including some option text or a list, and ending in a ; (semicolon).

The **named.conf** file is organized into stanzas. Each stanza is an enclosed set of configuration options that define either general characteristics of the daemon or a zone configuration. Certain stanza definitions are allowed only at the top-level, therefore nesting these stanzas is not allowed. The current top-level configuration stanza keywords are: acl, key, logging, options, server, and zone.

Further configuration information can be incorporated into the conf file via the **include** keyword. This keyword directs the daemon to insert the contents of the indicated file into the current position of the **include** directive.

Access Control List (ACL) Definition

```
acl acl-name {
    access-element;
    [ access-element; ... ]
};
```

Defines an access control list to be referenced thoughout the configuration file by*acl-name*. Multiple acl definitions can exist within one configuration file provided that each *acl-name* is unique. Additionally, four default access control lists are defined:

- **any** Any host is allowed.
- **none** No host is allowed.
- **localhost** Only the localhost is allowed.
- **localnets** Only hosts on a network matching a name server interface is allowed.

<table>
<tr><th>Option</th><th>Values</th><th>Explanation</th></tr>
<tr><td>access-element</td><td>IP-address
IP-prefix
acl-reference</td><td>Defines a source as allowed or disallowed. Multiple access-elements are allowed inside the acl stanza.

Each element can be an IP address in dot notation (e.g., 9.3.149.66) an IP prefix in CIDR or slash notation (e.g., 9.3.149/24) or a reference to another access control list (e.g., <code>localhost</code>).

Additionally, each element indicates whether the element is allowed or disallowed access via an ! (exclamation point) modifier prepended to the element.

For example:

<pre>acl hostlist1 {
    !9.53.150.239;
    9.3.149/24;
};</pre>
When the access control list "<code>hostlist1</code>" is referenced in the configuration, it implies to allow access from any host whose IP address begins with 9.3.149 and to disallow access from the internet host 9.53.150.239.</td></tr>
</table>

Key Definition

```
key key-name {
    algorithm alg-id;
    secret secret-string;
};
```

Defines an algorithm and shared secret key to be referenced in a server stanza and used for authentication by that name server. This feature is included for future use and is currently unused in the name server.

Option	Values	Explanation
algorithm	*alg-id* string	A quoted-string that defines the type of security algorithm that will be used when interpreting the secret string. None are defined at this time.
secret	*secret-string* string	A quoted-string that is used by the algorithm to authenticate the host.

Logging Configuration

```
logging {

    [ channel channel-name {
       ( file file-name
             [ versions ( num-vers | unlimited ) ]
             [ size size-value ]
       | syslog ( kern | user | mail | daemon |
                  syslog | lpr | news | uucp )
       | null );
       [ print-category ( yes | no ); ]
       [ print-severity ( yes | no ); ]
       [ print-time ( yes | no ); ]
      }; ... ]
    [ category category-name {
          channel-reference;
          [ channel-reference; ... ]
      }; ... ]
};
```

In this newest version of the name server, the logging facility has been greatly improved to allow for much reconfiguration of the default logging mechanism. The **logging** stanza is used to define logging output channels and to associate the predefined logging categories with either the predefined or user-defined logging output channels.

When no logging stanza is included in the conf file, the name server still logs messages and errors just as it has in previous releases. Informational and some critical messages will be logged through the syslog daemon facility, and debug and other esoteric information will be logged to the **named.run** file when the global debug level (set with the `-d` command-line option) is non-zero.

Option	Values	Explanation

channel		Defines an output channel to be referenced later by the *channel-name* identifier. An output channel specifies a destination for output messages to be sent as well as some formatting information to be used when writing the output message. More than one output channel can be defined provided that each *channel-identifier* is unique. Also, each output channel can be referenced from multiple logging categories. There are four predefined output channels: • **default_syslog** sends "info" and higher severity messages to syslog's "daemon" facility • **default_debug** writes debug messages to the **named.run** file as specified by the global debug level • **default_stderr** writes "info" and higher severity messages to stderr • **null** discards all messages
file	*file-name* string	Defines an output channel as one that logs messages to an output file. The file used for output is specified with the *file-name* string. Additionally, the **file** option allows for controlling how many versions of the output file should be kept, and what size limit the output file should never exceed. The **file**, **syslog**, and **null** output paths are mutually exclusive.
versions	*num-versions* unlimited	Specifies the number of old output files that should be kept. When an output file is reopened, rather than replacing a possible existing output file, the existing output file will be saved as an old output file with a **.***value* extension. Using the *num-versions* value, one can limit the number of old output files to be kept. However, specifying the `unlimited` keyword indicates to continually accumulate old output file versions. By default, no old versions of any log file are kept.

size	*size-value*	Specifies the maximum size of the log file used by this channel. By default, the size is unlimited. However, when a size is configured, once *size-valu*e bytes are written to the file, nothing more will be written until the file is reopened. Accepted values for *size-value* include the word "unlimited" and numbers with `k`, `m`, or `g` modifiers specifying kilobytes, megabytes, and gigabytes respectively. For example, `1000k` and `1m` indicate one thousand kilobytes and one megabyte respectively.
syslog	kern user mail daemon auth syslog lpr news uucp	Defines an output channel as one that redirects its messages to the syslog service. The supported value keywords correspond to facilities logged by the syslog service. Ultimately, the syslog service will define which received messages will be logged through the service, therefore, if definining a channel to redirect its messages to the syslog service's `user` facility would not result in any visibly logged messages if the syslog service is not configured to output messages from this facility. For more information concerning the syslog service, see the **syslogd** daemon. The **file**, **syslog**, and **null** output paths are mutually exclusive.
null		Defines an output channel through which all messages will be discarded. All other output channel options are invalid for an output channel whose output path is null.

severity	critical error warning notice info debug [*level*] dynamic	Sets a threshold of message severities to be logged through the output channel. While these severity definitions are similar to those used by the syslog service, for the name server they also control output through file path channels. Messages must meet or exceed the severity level to be logged through the output channel. The `dynamic` severity specifies that the name server's global debug level (specified when the daemon is invoked with the `-d` flag) controls which messages pass through the output channel. Also, the `debug` severity can specify a *level* modifier which is an upper threshold for debug messages whenever the name server has debugging enabled at any level. A lower debug level indicates less information is to be logged through the channel. It is not necessary for the global debug level to meet or exceed the debug *level* value. If used with the `syslog` output path, the syslog facility will ultimately control what severities are logged through the syslog service. For example, if the syslog service is configured to only log `daemon.info` messages, and the name server is configured to channel all debug messages to the syslog service, the syslog service will filter the messages from its output path.

<table>
<tr><td>print-category</td><td>yes
no</td><td rowspan="3">Controls the format of the output message when it is sent through the output path. Regardless of which, how many, or in which order these options are listed inside the channel stanza, the message will be prepended with the the text in a time, category, severity order.

The following is an example of a message with all three <code>print-</code> options enabled:

<code>28-Apr-1997 15:05:32.863 default: notice: Ready to answer queries.</code>

By default, no extra text will be prepended to an output message.

Note that when the syslog service logs messages, it also prepends the date and time information to the text of the message. Thus, enabling <code>print-time</code> on a channel that uses the syslog output path would result in the syslog service logging a message with two dates prepended to it.</td></tr>
<tr><td>print-severity</td><td>yes
no</td></tr>
<tr><td>print-time</td><td>yes
no</td></tr>
<tr><td>category</td><td></td><td>The <code>category</code> keyword defines a stanza which associates a logging or messaging category with predefined or user-defined output channels.

By default, the following categories are defined:

<code>category default { default_syslog; default_debug; };</code>
<code>category panic { default_syslog; default_debug; };</code></td></tr>
</table>

`category-name`	default config parser queries lame-servers statistics panic update ncache xfer-in xfer-out db event-lib packet notify cname security os insist maintenance load response-checks	The *category-name* specifies which logging category is to be associated with the listed *channel-references*. This results in any output text generated by the name server daemon for that logging category to be redirected through each of the *channel-references* listed. The `default` category defines all messages that are not listed in one of the specific categories listed. Also, the `insist` and `panic` categories are associated with messages that define a fatal inconsistency in the name server's state. The remaining categories define messages that are generated when handling specific functions of the name server. For example, the `update` category is used when logging errors or messages specific to the handling of a dynamic zone update, and the `parser` category is used when logging errors or messages during the parsing of the conf file.
channel-reference		References a *channel-name* identifier defined previously in the **logging** configuration stanza. Therefore, every message associated with the defined *category-name* will be logged through each of the defined *channel-references.*

Global Options

```
options {
    [ directory path-string; ]
    [ named-xfer path-string; ]
    [ dump-file path-string; ]
    [ pid-file path-string; ]
    [ statistics-file path-string; ]
    [ auth-nxdomain ( yes | no ); ]
    [ fake-iquery ( yes | no ); ]
    [ fetch-glue ( yes | no ); ]
    [ multiple-cnames ( yes | no ); ]
    [ notify ( yes | no ); ]
    [ recursion ( yes | no ); ]
    [ forward ( only | first ); ]
    [ forwarders { ipaddr; [...] }; ]
    [ check-names
       ( master|slave|response )
       ( warn|fail|ignore ); ]
    [ allow-query { access-element; [...] }; ]
    [ allow-transfer { access-element; [...] ); ]
    [ listen-on [ port port-num ] { access-element; [...] }; ... ]
    [ query-source [ address ( ipaddr|* ) ] [ port ( port|* ) ]; ]
    [ max-transfer-time-in seconds; ]
    [ transfer-format ( one-answer | many-answers ); ]
```

```
    [ transfers-in value; ]
    [ transfers-out value; ]
    [ transfers-per-ns value; ]
    [ coresize size-value; ]
    [ datasize size-value; ]
    [ files size-value; ]
    [ stacksize size-value; ]
    [ clean-interval value; ]
    [ interface-interval value; ]
    [ statistics-interval value; ]
    [ topology { access-element; [...] }; ]
};
```

Defines many globally available options to to modify basic characteristics of the name server.

Because some of the options in this configuration stanza may modify the behavior in how the **named** daemon will read and interpret later sections of the named file, it is highly recommended that the **options** stanza be the first stanza listed in the configuration file.

Option	**Values**	**Default**	**Explanation**

directory	*path-string*	`"."`	Indicates the directory from which all relative paths will be anchored. The *path-string* parameter must be a quoted string. For example, to indicate that all zone files will exist in the "/usr/local/named/data" without listing each file in the zone definitions, specify the global option directory as:
			`options {`
			`    directory`
			`"/usr/local/named/data";`
			`};`

named-xfer	*path-string*	`"/usr/sbin/named-xfer"`	Specifies the path and executable name of the **named-xfer** command used for inbound zone transfers. The *path-string* parameter must be a quoted string.
dump-file	*path-string*	`"/usr/tmp/named_dump.db"`	Specifies a filename to which the database in memory will be dumped whenever the **named** daemon receives a SIGINT signal.
pid-file	*path-string*	`"/etc/named.pid"`	Specifies the file in which the **named** daemon will write its PID value.
statistics-file	*path-string*	`"/usr/tmp/named.stats"`	Specifies the file to which the name server will append operating statistics when it receives the SIGILL signal.
auth-nxdomain	yes no	`yes`	Controls whether the server should respond authoritatively when returning an `NXDOMAIN` response.
fake-iquery	yes no	`no`	Controls whether the server should respond to the obsolete `IQUERY` requests.

fetch-glue	yes no	`yes`	Controls whether the server should search for "glue" records to include in the additional section of a query response.
multiple-cnames	yes no	`no`	Controls whether the server will allow multiple `CNAME` records for one domain name in any of its zone databases. This practice is discouraged but an option remains for backwards compatibility.
notify	yes no	`yes`	Controls whether the name server will send `NOTIFY` messages to its slave servers upon realization of zone changes. Because the slave servers will almost immediately respond to the `NOTIFY` message with a request for zone transfer, this limits the amount of time that the databases are out of synchronization in the master and slave relationship.
recursion	yes no	`yes`	Controls whether the server will attempt to resolve names outside of its domains on behalf of the client. If set to `no`, the name server will return a referral to the client in order for the client to continue searching for the name. Used with the `fetch-glue` option, one can contain the amount of data that grows in the name server's memory cache.
forward	only first	`first`	Controls how forwarding is used when forwarding is enabled. When set to `first`, the name server will attempt to search for a name whenever the forwarded host does not provide an answer. However, when set to `only`, the name server will not attempt this extra work.
forwarders	*ipaddr*	`(empty list)`	Enables the use of query forwarding when defining a Forwarding Name Server. The *ipaddr* parameter list specifies the hosts to which the query should be forwarded when it cannot be resolved from the local database. Each *ipaddr* is an internet address in standard dot notation.

check-names	master ignore
	master warn
	master fail
	slave ignore
	slave warn
	slave fail
	response ignore
	response warn
	response fail
		`master fail`
		`slave warn`
		`response ignore`
			Controls how the name server will handle non-RFC compliant host names and domain names through each of its operation domains.
			
			The `master` keyword specifies how to handle malformed names in a master zone file.
			The `slave` keyword specifies how to handle malformed names received from a master server.
			The `response` keyword specifies how to handle malformed names received in response to a query.
			
			`ignore` directs the server to ignore any malformed names and continue normal processing.
			`warn` directs the server to warn the administrator through logging, but to continue normal processing.
			`fail` directs the server to reject the name entirely. For the responses to queries, this implies that the server will return a `REFUSED` message to the original query host.

allow-query	*access-element*	`any`	Limits the range of querying hosts allowed to access the system. Each *access-element* is specified in the same manner as in the acl stanza defined earlier.
allow-transfer	*access-element*	`any`	Limits the range of querying hosts that are requesting zone transfers. Each *access-element* is specified in the same manner as in the acl stanza defined earlier.

listen-on	port *port-num*
	access-element
		`port 53 { localhost; }`	Limits the interfaces available to the name server daemon and controls which port to use to listen for queries. By default, the name server uses all interfaces on the system and listens on port 53. Additionally, multiple `listen-on` definitions are allowed within the **options** stanza.
			
			Each access element is specified in the same manner as in the acl stanza defined earlier. The following example limits the name server to using only the interface with address 9.53.150.239:
			
			`listen-on port 53 {`
			`9.53.150.239; };`

query-source	address *ipaddr* address * port *port* *port* *	`address * port *`	Modifies the default address and port from which queries will originate.

max-transfer-time-in	*seconds*	`120`	Specifies the maximum amount of time an inbound zone transfer will be allowed to run before it is aborted. This is used to control an event in which a child process of the name server does not execute or terminate properly.
transfer-format	one-answer many-answers	`one-answer`	Controls the method in which full zone transfers will be sent to requestors. The `one-answer` method uses one packet per zone resource record while `many-answers` will insert as many resource records into one packet as possible. While the `many-answers` method is more efficient, it is only understood by the newest revisions of the name server. This option can be overridden in the **server** stanza to specify the method on a per name server basis.
transfers-in	*value*	`10`	Specifies the maximum number of concurrent inbound zone transfers. While this will limit the amount of time each slave zone is out of synchronization with the master's database, because each inbound transfer runs in a separate child process, increasing the *value* may also increase the load on the slave server.
transfers-out	*value*	`N/A`	Specifies the maximum number of concurrent outbound zone transfers for the name server. This option is currently unused in the server, but will be available at a later time.
transfers-per-ns	*value*	`2`	Specifies the maximum amount of concurrent zone transfers from a specific remote name server. While this will limit the amount of time each slave zone is out of synchronization with the master's database, increasing this value may increase the load on the remote master server.
coresize	*size-value*	`default`	Configures some process specific values for the daemon. The default values or those inherited by the system and by the system's resources. Each *size-value* can be specified as a number or as a number followed by the `k`, `m`, and `g` modifiers indicating kilobytes, megabytes, and gigabytes respectively.
datasize	*size-value*	`default`	
files	*value*	`unlimited`	
stacksize	*size-value*	`default`	

clean-interval	*minutes*	`60`	Controls the intervals for the periodic maintenance tasks of the name server. The `clean-interval` specifies how frequently the server will remove expired resource records from the cache. The `interface-interval` specifies how frequently the server will rescan for interfaces in the system. The `statistics-interval` specifies how frequently the name server will output statistics data. A *minutes* value of zero indicates that the service task should only run when the configuration file is reread.
interface-interval	*minutes*	`60`	
statistics-interval	*minutes*	`60`	
cleandb-time	*time*	`N/A`	Specifies a time of day in which the database will be scanned and any dynamic records whose set of `SIG` resource records are all expired will be removed. For a dynamic zone which has `update-security` set to `presecured`, only the expired `SIG KEY` will remain. The default is to never perform this scan. Instead, the expired records will remain until the name is queried. *time* is specified as `HH:MM` in a 24-hour format.
topology	*access-element*	`localhost; localnets;`	Specifies a search order to use to find a preference in a list of addresses corresponding to a name server. Whenever a query is forwarded or a query must be made to another name server, it may be necessary to choose an address from a list of available addresses. Each *access-element*, while seemingly similar to those specified in an acl stanza, is interpretted by its position in the list. The first elements in the list are preferred more than those following them. Negated elements (those specified with the ! (exclamation point) modifier) are considered least desirable.

Server Specific Options

```
server ipaddr
{
    [ bogus ( yes | no ); ]
    [ transfers value;
]
    [ transfer-format ( one-answer |
many-answers ); ]
}
```

Modifies the behavior in which the remote name server matching the specified *ipaddr* IP address should be treated.

Option	Values	Explanation
bogus	yes no	Indicates that the name server identified by the stanza should not be used again. The default value is `no`.
transfers	*value*	Overrides the globally available option `transfers-per-ns`. Specifies a maximum value for the number of concurrent inbound zone transfers from the foreign name server identified by the stanza.
transfer-format	one-answer many-answers	Overrides the globally available option `transfer-format` to a specific value for the specified server. The `transfer-format` option indicates to the name server how to form its outbound full zone transfers. By default, the value is inherited from the options stanza (where it defaults to `one-answer`). `one-answer` specifies that only one resource record can be sent per packet during the zone transfer, whereas `many-answers` indicates to entirely fill the outbound packet with resource records. The `many-answers` format is only available in the newest revisions of the name server.

Zone Definition

```
zone domain-string [ class ] {
    type ( hint | stub | slave | master );
    [ file path-string; ]
    [ masters { ipaddr; [...] }; ]
    [ check-names ( warn | fail | ignore ); ]
    [ allow-update { access-element; [...] }; ]
    [ update-security ( unsecured | presecured | controlled ); ]
    [ allow-query { access-element; [...] }; ]
    [ allow-transfer { access-element; [...] }; ]
    [ max-transfer-time-in seconds; ]
    [ notify ( yes | no ); ]
    [ also-notify { ipaddr; [...] }; ]
    [ dont-notify { ipaddr; [...] }; ]
    [ notify-delaytime seconds; ]
    [ notify-retrytime seconds; ]
    [ notify-retrycount value; ]
    [ dump-interval seconds; ]
    [ incr-interval seconds; ]
    [ deferupdcnt value; ]
    [ key-xfer ( yes | no ); ]
    [ timesync ( yes | no ); ]
    [ timesync-xfer ( yes | no ); ]
```

```
    [ save-backups ( yes | no ); ]
    [ ixfr-directory path-string; ]
    [ separate-dynamic ( yes | no ); ]
};
```

The zone stanza is used to define a zone, its type, possible location of data, and operating parameters. The *domain-string* is a quoted string specifying the zone, where "." is used to specify the root zone. The *class* paramter specifies the *class* of the zone as either `in`, `hs`, `hesiod`, or `chaos`. By default, the *class* is assumed to be `IN`.

Option	Values	Default	Explanation
type	hint stub slave master	`N/A`	Defines the type of the zone. `hint` zones, previously regarded as cache zones, only describe a source for information not contained in the other defined zones. A `stub` zone is one similar to a `slave` zone. While the `slave` zone replicates the entire database of its master, the `stub` zone only replicates the `NS` resource records. The `master` zone maintains a database on disk. Based upon the selection of zone type, some of the other options are required while others may be impertinent. Zones of type `hint` and `master` require the `file` option, while zones of type `slave` and `stub` require the `masters` option. Additionally, the only other option available to a hint zone is the `check-names` option.
file	*path-string*	`N/A`	Specifies the location for the source of data specific to the zone. This parameter is only optional for `stub` and `slave` zones, where its inclusion indicates that a locally saved copy of the remote zone can be kept. The *path-string* parameter is a quoted string which can specify the file name either non-relative or relative to the `options` stanza's `directory`. If the path is intended to be specified relative to the server root, the `options` stanza must be specified before the zone stanza.
masters	*ipaddr*	`N/A`	Specifies a list of sources that will be referenced for a `slave` or `stub` zone to retrieve its data. This option is not valid for any other type of zone, and must be included for either of these two types.

check-names	warn fail ignore		Overrides the `check-names` option in the global `options` stanza. The default value is inherited from the `options` stanza, where its default is `fail` for `master` zones and `warn` for `slave` zones.
allow-update	*access-element*	`none`	Indicates from what source addresses a zone will accept dynamic updates. *access-elements* are specified in the same manner as they are for the `acl` stanza. Because of the inherint insecurity of a dynamic update, this value defaults to `none`. If no `update-security` is specified, dynamic updates should be limited to a specific set of secured machines.
update-security	unsecured presecured controlled	`unsecured`	Valid only when the `allow-update` option specifies at least one source address, `update-security` defines what type of secured update mechanism the zone will use. The current zone update security method is a non-standard two-key method, but is compatible with previous releases of the name server. `presecured` indicates that a zone will only accept updates for which names and resource records already exist, unless the update is signed by the zone's authorizing key. Normally, this means that the zone must be prepopulated with the names and records it is to maintain. `controlled` specifies a zone in which names can be added to the database without the signature of the zone's authorizing key, but existing records cannot be modified without being signed by the `KEY` resource record's corresponding private key. Note that a proper `presecured` or `controlled` zone must contain a zone `KEY` resource record. See the TCP/IP Name Resolution for more information regarding zone update security.
allow-query	*access-element*		Overrides the globally available option `allow-query`. This option's default is inherited from the global `options` stanza, where its default is `any`.

allow-transfer	*access-element*		Overrides the globally available option `allow-transfer`. This option's default is inherited from the global `options` stanza, where its default is `any`.
max-transfer-time-in	*seconds*		Overrides the globally available option `max-transfer-time-in`. This option's default is inherited from the global `options` stanza, where its default is `120`.
notify	yes no		Overrides the globally available option `notify`. This option's default is inherited from the global `options` stanza, where its default is `yes`.
also-notify	*ipaddr*	`N/A`	The default `NOTIFY` mechanism will notify slave servers of a change in the DOMAIN database in order to limit the amount of time that the slave server retains a zone out of synchronization with the master server. The `also-notify` option allows for the addition of addresses to submit the notifications.
dont-notify	*ipaddr*	`N/A`	Specifies a list of IP addresses to be removed from the default list of `NOTIFY` recipients. This option is useful if a name server is known to be problematic when receiving `NOTIFY` requests.
notify-delaytime	*seconds*	`30`	Specifies an estimated time of delay between notifications to multiple name servers. Because the receipt of a `NOTIFY` message usually triggers the prompt request for a zone transfer, this option can tune to latency in which each server will respond with the request for the modified zone. The real value used will be randomized between the specified number of *seconds* and twice this value.
notify-retrytime	*seconds*	`60`	Specifies the number of *seconds* in which the name server will wait to retransmit a `NOTIFY` message which has gone unresponded.
notify-retrycount	*value*	`3`	Specifies the maximum number of tries that the name server will attempt to send unanswered `NOTIFY` messages to other name servers.

dump-interval	*seconds*	`3600`	Specifies an interval in which the name server will rewrite a dynamic zone to the zone `file`. In the interim, all updates and other transactions will be logged in the transaction log file for performance reasons. Aside from this periodic zone dump, the transaction log file is only discarded and the zone is only dumped when the name server is properly shut down. This option is only valid for zones in which the `allow-update` option specifies at least one valid accessor. Note: The transaction log file name is the zone file name with an appended "`.log`" extension.
incr-interval	*seconds*	`300`	Specifies an interval in which the name server will accept dynamic updates while not increasing the zone's `SOA` record's serial level. Because a change in the zone `SOA` record will instantiate a `NOTIFY` message, limiting this occurrence will limit the amount of zone transfer requests at the expense of minimal zone differences between a dynamic master server and its slave. This option is only valid for zones in which the `allow-update` option specifies at least one valid accessor.
deferupdcnt	*value*	`100`	Specifies a threshold value for the number of properly applied updates received during one `incr-interval` interval. If more than *value* updates are realized during the interval, the name server will modify the zone SOA serial level and subsequently NOTIFY each of the slave servers. Use this value to limit the database replication inconsistencies in an environment where dynamic zone updates occur infrequently but in large magnitude. This option is only valid for zones in which the `allow-update` option specifies at least one valid accessor.

key-xfer	yes no	`yes`	Specifies whether the server should transmit `KEY` resource records during a zone transfer. In a very controlled environment where `KEY` queries will only be made to the master name server, setting this option to `no` will save zone transfer time and improve performance.
timesync	yes no	`yes`	Specifies that a name server should calculate the true expiration time of a `SIG` resource record using its own clock rather than relying on the expiration time set by a possible update source. This removes the inconsistencies involved when dynamic zone updaters have their system clocks misaligned from the name server host. Because enabling this option modifies the output and interpretation of a `SIG` resource record in a `DOMAIN` database file, disabling this option may be required when manually transfering a `DOMAIN` database file to another name server.
timesync-xfer	yes no	`yes`	Specifies which `SIG` resource record expiration time will be transfered during a zone transfer. Enabling this option is only valid when the `timesync` option is enabled.

ixfr-directory	*path-string*		Specifies a directory in which temporary data files will be contained for use with this zone. The datafiles contain incremental zone changes and are essential to the proper use of the Incremental Zone Transfer (`IXFR`) method. Because these files are created and destroyed dynamically by the name server, one should not specify a globally-writable directory. Additionally, the directory specified must be unique from other `ixfr-directory` options specified in other zones. The default value for this directory is derived from the zone's `file` name or domain name. By default, a directory is created in an "`ixfrdata`" directory within the name server's default directory. Contained in this directory will be subdirectory matching the base name of the zone's `file` name or domain name. It is not necessary to specify this option for the proper behavior of the `IXFR` feature.
save-backups	yes no	`no`	To properly calculate an incremental zone difference between server invocations, it is necessary to determine the zone database differences prior to the shutdown of the server and after the loading of the server. By enabling this option, a backup of the zone file will be written and read upon loading of the name server to determine any zone differences. While enabling this option is necessary to use the IXFR transfer method after a stop and restart transition of the name server, it is not necessary to realize incremental zone differences when a zone file is modified and signalled to reload via the SRC **refresh** command or `SIGHUP` signal.

separate-dynamic	yes no	no	Instructs the name server to retain `$INCLUDE` references in a dynamic zone when the `DOMAIN` database file is written to disk. The behavior of this feature implies that resource records that can be modified through the dynamic update mechanism exist in the `DOMAIN` database file referenced by the `file` option, while other resource records that should not be modified through the dynamic update mechanism be contained in files included (through the `$INCLUDE` directive) by the `DOMAIN` database file.

Examples

The following examples show the some of the various ways to use configure a simple **named.conf** file. In these examples, two networks are represented: `abc` and `xyz`.

Network `abc` consists of:

- `gobi.abc`, the master name server for the `abc` network, 192.9.201.2
- `mojave.abc`, a host machine, 192.9.201.6
- `sandy.abc`, a slave name server for the `abc` network and the gateway between `abc` and `xyz`, 192.9.201.3

Network `xyz` consists of:

- `kalahari.xyz`, master 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 hint name server for the `xyz` network, 160.9.201.13
- `sandy.xyz`, a slave 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 a slave name server for both domains.

1. The **/etc/named.conf** file for `gobi.abc`, the master name server for network `abc`, contains these entries:

```
#
# conf file for abc master server  - gobi.abc
#
server 192.9.201.3 {
   transfer-format many-answers;
};
zone "abc" in {
   type master;
   file "/etc/named.abcdata";
```

```
    allow-update { localhost; };
};
zone "201.9.192.in-addr.arpa" in {
    type master;
    file "/etc/named.abcrev";
    allow-update { localhost; };
};
zone "0.0.127.in-addr.arpa" in {
    type master;
    file "/etc/named.abclocal";
};
```

2. The **/etc/named.conf** file for `kalahari.xyz`, the master name server for network `xyz`, contains these entries:

```
#
# conf file for abc master server  -  kalahari.xyz
#
acl xyz-slaves {
    160.9.201.3;
};
options {
    directory "/etc";
    allow-transfer { xyz-slaves; localhost; };
};
zone "xyz" in {
    type master;
    file "named.xyzdata";
};
zone "9.160.in-addr.arpa" in {
    type master;
    file "named.xyxrev";
};
zone "0.0.127.in-addr.arpa" in {
    type master;
    file "named.xyzlocal";
};
```

3. The **/etc/named.conf** file for `sandy`, the slave name server for networks `abc` and `xyz`, contains the following entries:

```
#
# conf file for slave server for abc and xyz - sandy
#
options {
    directory "/etc";
};
zone "abc" in {
    type slave;
    masters { 192.9.201.2; };
    file "named.abcdata.bak";
};
zone "xyz" in {
    type slave;
```

```
        masters { 160.9.201.4; };
        file "named.xyzdata.bak";
    };
    zone "201.9.192.in-addr.arpa" in {
        type slave;
        masters { 192.9.201.2; };
    };
    zone "9.160.in-addr.arpa" in {
        type slave;
        masters { 192.9.201.4; };
    };
    zone "0.0.127.in-addr.arpa" in {
        type master;
        file "named.local";
    };
```

4. The **/etc/named.conf** file for `sahara`, a hint name server for the network `xyz`, contains the following entries:

```
    #
    # conf file for hint server for xyz - sahara
    #
    zone "." in {
        type hint;
        file "/etc/named.ca";
    };
    zone "0.0.127.in-addr.arpa" in {
        type master;
        file "/etc/named.local";
    };
```

Files

/usr/samples/tcpip/named.conf	Contains the sample **named.conf** file.

Related Information

The **named** daemon.

The **syslogd** daemon.

The **DOMAIN cache** file format, **DOMAIN local** file format, **DOMAIN data** file format, **DOMAIN Reverse data** file format, **rc.tcpip** file format.

Configuring a Primary Name Server and Naming

.netrc File Format for TCP/IP

Purpose

Specifies automatic login information for the **ftp** and **rexec** commands.

Description

The **$HOME/.netrc** file contains 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 **securetcpip** command is running on your system.

The **.netrc** can contain the following entries (separated by spaces, tabs, or new lines):

machine *HostName*	The *HostName* variable is the name of a remote host. This entry begins the definition of the automatic login process for the specified host. All following entries up to the next machine entry or the end of the file apply to that host.
default	The *default* variable is the same as *machine* except that *default* matches any name. There can be only one default entry. It must be the last entry (after all machine entries); otherwise, entries that follow it will be ignored. This is normally used as: `default login anonymous password user@site` thereby giving the user automatic anonymous ftp login to machines not specified in the **.netrc** file. This can be overridden by using the **-n** flag to disable the auto-login.
login *UserName*	The *UserName* variable is the full domain user name for use at the remote host. If this entry is found, the automatic login process initiates a login, using the specified name. If this entry is missing, the automatic login process is unsuccessful.
password *Password*	The *Password* variable is the login password to be used. The automatic login 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 the **.netrc** file. Otherwise the automatic login process is unsuccessful, 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 automatic login process supplies the password to the remote server. If the remote host requires an account password but this entry is missing, the automatic login 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 automatic login process. The **rexec** command does not recognize a **macdef** entry.

Examples

The following is an example of an entry in a **.netrc** file:

```
machine host1.austin.century.com login fred password bluebonnet
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/netrc	Contains a sample **.netrc** file with directions for its use.

Related Information

The **ftp** command, **rexec** command, **securetcpip** command.

networks File Format for TCP/IP

Purpose

Contains 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	Specifies an official network name.
Number	Specifies a network number.
Aliases	Specifies any unofficial names used for the network.

Items on a line are separated by one or more spaces 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 can be modified locally to include unofficial aliases or unknown networks.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/networks	Contains a sample **networks** file, which also contains directions for its use.

Related Information

The **routed** daemon.

The **getnetent** subroutine.

nroff or troff Input File Format

Purpose

Specifies input file format for the **nroff** and **troff** commands.

Description

The **nroff** and **troff** commands format text for printing by interspersing the text with control sequences. Control sequences are either control line requests or escape requests that control text processing by the printing device.

Control lines begin with a control character followed by a one- or two-character name that specifies a basic request or a user-defined macro. Default control characters are the . (dot) or the ' (apostrophe). The ' (apostrophe) control character suppresses the **nroff** or **troff** command break function, which is caused by some requests. This break function forces output of a partially filled line. To separate the control character from the request or macro, use white space created with either a tab or the space bar. The **nroff** and **troff** commands ignore control lines with unrecognized names.

Escape requests can be inserted anywhere in the input text by means of an escape character. The \ (backslash) character is the default escape character. For example, the escape request `\nr` causes the contents of the number register, `r`, to be read.

> **Note:** If text must begin a line with a . (dot), a zero-width character sequence (\&) must precede the control character. This is true even if the control character is preceded by an escape request. The zero-width character prevents the command from interpreting the text as a control character. See the example for an illustration of the use of a zero-width character.

Examples

To print the words `.dean`, enter:

```
\fB\&.dean
```

If you neglected to add the `\&`, the formatter would read the statement as the macro request:

```
.de an
```

Implementation Specifics

This file is part of the Formatting Tools in the Text Formatting System.

Related Information

The **nroff** command, **troff** command.

nterm File Format

Purpose

Describes 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/share/lib/nterm/tab.***Name* file, where the *Name* variable is the name for a terminal type.

The first line of a driving table should contain the name of the terminal, which is simply a string with no imbedded white space (any combination of spaces, tabs, and newline characters). 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*]
- **codeset** [*Character String*]

The meanings of these fields are as follows:

bset
: Specifies bits to set in the `c_oflag` field of the **termio** structure before output.

breset
: Specifies bits to reset in the `c_oflag` field of the **termio** structure before output.

hor
: Specifies horizontal resolution in units of 1/240 of an inch.

vert
: Defines vertical resolution in units of 1/240 of an inch.

newline
: Defines space moved by a new-line (linefeed) character in units of 1/240 of an inch.

char
: Defines a quantum of character sizes, in units of 1/240 of an inch (that is, a character is a multiple of **char** units wide).

em
: Defines the size of an em space in units of 1/240 of an inch.

halfline
: Defines the amount of space moved by a half-linefeed (or half-reverse-linefeed) character in units of 1/240 of an inch.

adj
: Defines a 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
: Specifies a sequence of characters used to initialize the terminal in a mode suitable for the **nroff** command.

twrest
: Specifies a sequence of characters used to restore the terminal to normal mode.

twnl
: Specifies a sequence of characters used to move down one line.

hlr
: Specifies a sequence of characters used to move up one-half line.

hlf
: Specifies a sequence of characters used to move down one-half line.

flr
: Specifies a sequence of characters used to move up one line.

bdon
: Specifies a sequence of characters used to turn on hardware boldface mode, if any.

bdoff
: Specifies a sequence of characters used to turn off hardware boldface mode, if any.

iton
: Specifies a sequence of characters used to turn on hardware italics mode, if any.

itoff
: Specifies a sequence of characters used to turn off hardware italics mode, if any.

ploton
: Specifies a sequence of characters used to turn on hardware plot mode (for Diablo-type mechanisms), if any.

plotoff
: Specifies a sequence of characters used to turn off hardware plot mode (for Diablo-type mechanisms), if any.

up
: Specifies a sequence of characters used to move up one resolution unit (**vert**) in plot mode, if any.

down
: Specifies a sequence of characters used to move down one resolution unit (**vert**) in plot mode, if any.

right
: Specifies a sequence of characters used to move right one resolution unit (**hor**) in plot mode, if any.

left
: Specifies a sequence of characters used to move left one resolution unit (**hor**) in plot mode, if any.

codeset *CodeSetName*
: Specifies the code set for the particular output device. *CodesetName* is any valid name for use with the **iconv** command. The code set defines character entries within the font description file for the character set section. The code set field is optional. If used, the code set field must follow the "left" field and precede the character set section, if provided. The default is **IBM-850**.

  The **nroff** command uses the specified *CodesetName* and the code set implied by the current locale to determine if code set conversions are necessary for the input characters. The **iconv** function is used to perform the code set conversion if necessary.

This part of the driving table is fixed-format; you cannot change the order of entries. Entries should be on separate lines each containing 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 only 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 2-character names, such as `\(hy`. If the **nroff** command does not find the word `charset` where it expects, it terminates processing with an error message.

Each definition after `charset` occupies one line and has the following format:

```
chname width output
```

The `chname` field is the (2-letter) name of the special character, the `width` field is its width in ems, and the `output` field is the string of characters and escape sequences to send to the terminal to produce the special character.

International Character Support

For fonts for large character sets in which most characters are the same width, as in Japanese, Chinese, and Korean, prototype characters are provided for the character set section of the **nterm** table. These prototype characters specify the width of characters of varying byte lengths. The code field for prototype character entries must contain a single **?** (question mark). The prototype character entries apply to all characters not explicitly defined on their own in the character set section. It is assumed the output device code for characters handled via prototype characters is the same as the input code for characters (with possible codeset conversions). The following are the prototype character definitions:

X1	Width	?	Represents the width of all one-byte characters not defined elsewhere.
X2	Width	?	Represents the width of all two-byte characters not defined elsewhere.
X3	Width	?	Represents the width of all three-byte characters not defined elsewhere.
X4	Width	?	Represents the width of all four-byte characters not defined elsewhere.

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/share/lib/nterm** directory. Then, give it the name **tab**.*xyz*, where the *xyz* variable is the name of the terminal and also the name that you pass the **nroff** command by way of the **-T** flag. For example:

```
nroff -Txyz
```

Implementation Specifics

This file is part of Formatting Tools in the Text Formatting System.

Files

/usr/share/lib/nterm/tab.*Name*	Contains terminal files.

Related Information

The **iconv** command, **nroff** command.

Permissions File Format for BNU

Purpose

Specifies BNU permissions for remote systems that call or are called by the local system.

Description

The **/etc/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 the local system contacts using BNU. These entries correspond to entries in the **/etc/uucp/Systems** file or other systems files listed in the **/etc/uucp/Sysfiles** file with the same format. 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 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 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. 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.

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.

Attention: Access permissions set in the **Permissions** file affect all BNU communications, including those made through the mail facility or over a TCP/IP connection. Entries in a **Permissions** file do *not* affect a remote-system user with a valid login on a specified local system. Remote login commands (such as **cu**, **ct**, **tn**, or **tip**) connect to and log in on a system regardless of the restrictions 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).

Note: Examples of using the **Permissions** file are provided . The examples include issuing default or restricted access to remote systems and combining LOGNAME and MACHINE entries.

LOGNAME and MACHINE Entries

The **Permissions** file contains two types of required entries:

LOGNAME
Specifies the login IDs and access permissions for remote systems that are allowed to contact the local system.

MACHINE
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*. LOGNAME entries take effect when a remote system contacts the local system. 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, if remote system `hera` contacts local 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 most restrictive LOGNAME and MACHINE 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 default permissions. To override these defaults, include option/value pairs in the entry. The available options are:

- **REQUEST**
- **SENDFILES**
- **READ**, **WRITE**
- **NOREAD**, **NOWRITE**
- **COMMANDS**
- **VALIDATE**
- **CALLBACK**

These options allow different remote systems different types of access to the local system when using the BNU file transport and command execution programs. A LOGNAME and a MACHINE entry can be combined into a single entry when both include the same options.

LOGNAME Entry

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. The format of a LOGNAME entry is:

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 **/etc/uucp/Systems** file or an alternative **uucico** service systems file specified in **/etc/uucp/Sysfiles** on the local system.

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. The remote system's access to the local system is then defined by these 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 (**/var/spool/uucppublic/***Syste mName*) 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.

> **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.

> **Attention:** Allowing remote systems to log in 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. Each remote system that contacts the local system should have a unique login ID with a unique UID.

MACHINE Entry

The **Permissions** file contains a MACHINE entry for each remote system the local system is permitted to contact. 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. 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:

MACHINE=*SystemName*[**:***SystemName* . . .]

The system names are separated by a colon. The entry includes no spaces or tab characters. 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 LOGNAME entry.

The *SystemName* list in a MACHINE entry may include a number of different remote systems. A MACHINE entry can also be:

MACHINE=OTHER [*Option***=***Value* . . .]

where the word OTHER represents a system name. This sets up access permissions for remote systems not specified in the existing MACHINE entries in a **Permissions** file. The **MACHINE=OTHER** entry is useful in these circumstances:

- When your installation includes a large number of remote systems that the local system regularly contacts for **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 create 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.

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.

Option/Value Pairs

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.

CALLBACK Option

The CALLBACK option, included in LOGNAME entries, specifies that no file transfer transactions will occur until the local system contacts the targeted remote system. The format of the CALLBACK option is either:

CALLBACK=no

OR

CALLBACK=yes

Note: 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, use the **CALLBACK=yes** option to specify that the local system must contact the remote system before the remote system may transfer any files to the local system.

If you include the **CALLBACK=yes** option in the LOGNAME entry, you must also have a MACHINE entry for that system so that your system can call it back. 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 COMMANDS option, included only in a MACHINE entry, specifies the commands that the remote systems listed in that MACHINE entry can execute on the local system. The format of the COMMANDS option is either:

COMMANDS=*CommandName*[**:***CommandName* . . .]

OR

COMMANDS=ALL

The default is **COMMANDS=rmail:uucp**. Under the default, remote systems can run only the **rmail** and **uucp** commands on the local system. (Users enter the **mail** command, which then calls the **rmail** command.)

The commands listed in the COMMANDS option 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 **/usr/bin** directory.

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.

Attention: The COMMANDS option can jeopardize the security of your system. Use it with extreme care.

NOREAD and NOWRITE Options

The NOREAD and NOWRITE options, 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 READ and WRITE options, 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. You can specify more than one path for **uucico** daemon activities.

The default location for both the READ and WRITE options is the **/var/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.

If the READ and WRITE options are not present in the **Permissions** file, the BNU program transfers files only to the **/var/spool/uucppublic** directory. However, if you specify path names in these options, enter the path name for every source and destination, including the **/var/spool/uucppublic** directory if the remote system is to be permitted access to it.

> **Attention:** Specifications with the READ, WRITE, NOREAD, and NOWRITE options affect the security of your local system in terms of BNU transactions. 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 REQUEST option, used in both LOGNAME and MACHINE entries, enables 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, the remote system may also request permission to receive any files queued on the local system for transfer to or execution on that remote system. This format of the REQUEST option permits such requests:

REQUEST=yes

The default, **REQUEST=no**, does not have to be entered. This specifies that the remote system cannot ask to receive any work queued for it on the local system. The local system must contact the remote system before transmitting files and execute commands queued on the local system to the remote system.

Use the **REQUEST=yes** option in both LOGNAME and MACHINE entries to allow remote-system users to transfer files to and execute commands on a local system on demand. Restrict 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 default allows 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 following SENDFILES option:

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.

> **Notes:**
> 1. The SENDFILES option is ignored when it is included in a MACHINE entry.
> 2. 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 VALIDATE option provides more security when including commands in the default command set that could cause damage when executed by a remote system on a local system. Use this option, specified only in a MACHINE entry, in conjunction with a COMMANDS option. The format of the VALIDATE option is:

VALIDATE=*LoginName*[**:** *LoginName* . . .]

The VALIDATE option verifies the identity of the calling remote system. Including this option in a MACHINE 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. Therefore, the **uuxqt** daemon 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 `/var/spool/uucppublic/hera` directory on system `zeus`.

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 (`/var/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.

Security

Access Control: Only a user with root authority can edit the **Permissions** file.

Examples

The following are examples of using the **Permissions** file.

Providing Default Access to Remote Systems

1. To provide the default permissions to any system logging in as `uucp1`, enter:

   ```
   LOGNAME=uucp1
   ```

2. To provide the default permissions to systems `venus`, `apollo`, and `athena` when called by the local system, enter:

   ```
   MACHINE=venus:apollo:athena
   ```

Providing Less Restricted Access to Remote Systems

1. The following LOGNAME entry allows remote system `merlin` to read and write to more directories than just the spool directory:

   ```
   LOGNAME=umerlin READ=/ NOREAD=/etc:/usr/sbin/uucp
   WRITE=/home/merlin:/var/spool/uucppublic
   ```

   A system logging in as user `umerlin` can read all directories except the **/usr/sbin/uucp** and **/etc** directories, but can write only to the **/home/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.

2. The following example allows remote system hera unrestricted access to system zeus, and shows the relationship between the LOGNAME and MACHINE entries:

```
LOGNAME=uhera REQUEST=yes SENDFILES=yes READ
=/ WRITE=/MACHINE=hera VALIDATE=uhera REQUEST=yes \COMMANDS=ALL READ=/ WRITE=/
```

   The remote system hera may engage in the following **uucico** and **uuxqt** transactions with 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 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.

Attention: This entry allows unrestricted access to the local system by the remote system listed in the MACHINE entry. This entry can jeopardize the security of your system.

Combining LOGNAME and MACHINE Entries

1. Following are LOGNAME and MACHINE entries for system hera:

```
LOGNAME=uhera REQUEST=yes SENDFILES=yes
MACHINE=hera VALIDATE=uhera 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:

```
LOGNAME=uhera SENDFILES=yes REQUEST=yes \
MACHINE=hera VALIDATE=uhera COMMANDS=rmail:news:uucp
```

2. 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

To allow your system to call systems that 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

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:/etc/uucp \
NOWRITE=/usr/secure:/etc/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=/var/spool/uucppublic:/home/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/sbin/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 **/home/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 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp/Permissions file	Describes access permissions for remote systems.
/etc/uucp/Systems file	Describes accessible remote systems.
/etc/uucp/Sysfiles file	Specifies possible alternative files for the **/etc/uucp/Systems** file.
/var/spool/uucppublic directory	Contains files that have been transferred.

Related Information

The **chmod** command, **mail** command, **rmail** command, **uucheck** command, **uucpadm** command.

The **uucico** daemon and **uuxqt** daemon read the **Permissions** file.

Configuring BNU, Understanding the BNU File and Directory Structure, Understanding BNU Security in *AIX Version 4.3 System Management Guide: Communications and Networks*.

phones File Format for tip

Purpose

Describes connections used by the **tip** command to contact remote systems.

Description

The **/etc/phones-file** file lists the remote systems that can be contacted using the **tip** command, along with the telephone numbers used to contact those systems.

A sample **phones-file** file for the **tip** command is included with the operating system. The sample file is named **/usr/lib/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** file. The individual phones file can be named with any operating system file name and placed in any directory to which the user has access. To instruct the **tip** command to use the new file, either 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** file, in the file specified by the **REMOTE** environment variable, or in the file specified by the **tip** command **remote** variable.

Format of Entries

The format of an entry in the **phones** file is:

```
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 beginning 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/phones	Denotes complete path name of the **phones** file.
/usr/lib/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.

Poll File Format for BNU

Purpose

Specifies when the BNU program should poll remote systems.

Description

The **/etc/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 **/var/spool/cron/crontabs/uucp** file, **uudemon.hour** command, and **uudemon.poll** command. Together, these files are responsible for initiating automatic calls to certain remote systems.

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. Modify the times specified in the **Poll** file based on how the systems at your site are used. 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.

Notes:

1. Only someone with root user authority can edit the **Poll** file, which is owned by the **uucp** program login ID.
2. Most versions of UUCP require a tab character between the *SystemName* field and the first *Time* field. In BNU, either a tab or spaces will work.

Examples

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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/locks	Contains lock files that prevent multiple uses of devices and multiple calls to systems.
/var/spool/cron/crontabs/uucp	Schedules BNU jobs for the **cron** daemon.

Related Information

The **uucpadm** command, **uudemon.hour** command, **uudemon.poll** command.

The **cron** daemon.

Configuring BNU, Setting Up BNU Polling of Remote Systems, Understanding the BNU File and Directory Structure in *AIX Version 4.3 System Management Guide: Communications and Networks*.

profile File Format

Purpose

Sets the user environment at login time.

Description

The **$HOME/.profile** file contains commands that the system executes when you log in. The **.profile** also provides variable profile assignments that the system sets and exports into the environment. The **/etc/profile** file contains commands run by all users at login.

After the **login** program adds the **LOGNAME** (login name) and **HOME** (login directory) variables to the environment, the commands in the **$HOME/.profile** file are executed, if the file is present. The **.profile** file contains the individual user profile that overrides the variables set in the **profile** file and customizes the user-environment profile variables set in the **/etc/profile** file. 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.

> **Note:** The **$HOME/.profile** file is used to set environments for the Bourne and Korn shells. An equivalent environment for the C shell is the **$HOME/.cshrc** file.

Examples

The following example is typical of an **/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=/usr/bin:/usr/sbin:/etc::
#Set terminal type
TERM=lft
#Make some environment variables global
export MAIL PATH TERM
```

Implementation Specifics

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

Files

/etc/profile	Contains profile variables.

Related Information

The **bsh** command, **csh** command, **env** command, **login** command, **mail** command, **mkuser** command, **ksh** command, **stty** command, **su** command.

The Profiles Overview in *AIX Version 4.3 System Management Concepts: Operating System and Devices* discusses profiles and how they can be modified for individual needs.

The Shells Overview in *AIX Version 4.3 System User's Guide: Operating System and Devices* describes what shells are, the different types, and how they affect the way commands are interpreted.

protocols File Format for TCP/IP

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 corresponds to the form:

Name Number Aliases

The fields contain the following information:

Name	Specifies an official Internet Protocol name.
Number	Specifies a protocol number.
Aliases	Specifies any unofficial names used for the protocol.

Items on a line are separated by one or more spaces 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 Base Operating System (BOS) Runtime.

Related Information

The **getprotoent** subroutine.

queuedefs File Format

Purpose

Specifies the handling of **cron** daemon events.

Description

The **/var/adm/cron/queuedefs** file defines how the system handles different **cron** daemon events types. The file specifies the maximum number of processes per event type to schedule at one time, the nice value of the event type, and how long to wait before retrying to execute a process. The following event types can be scheduled by the **cron** daemon:

- **at** command events
- **batch** command events
- **crontab** command events
- **sync** subroutine events
- **ksh** command events
- **csh** command events

This file is empty as shipped, but can be modified to change how the **cron** daemon handles each event type. Each entry in the **queuedefs** file is of the form:

EventType.[Jobsj][Nicen][Waitw]

The fields are described as follows:

EventType	Specifies a character representing an event type. The following are valid values for the *EventType* field:
a	Specifies an **at** command event.
b	Specifies a **batch** command event.
c	Specifies a **crontab** command event.
d	Specifies a **sync** subroutine event.
e	Specifies a **ksh** command event.
f	Specifies a **csh** command event.
Jobsj	Specifies the maximum number of jobs the **cron** daemon can start at one time. The default value is 100.
Nicen	Specifies the nice value for job execution. The default value is 2.
Waitw	Specifies the time, in seconds, to wait before attempting to execute the command again. The default value is 60 seconds.

Note: You must have root user authority to modify this file.

The **at** command allows you to specify the time when a command should be run. Each command or program will be assigned a job number and will be queued in the **/var/spool/cron/atjobs** directory.

The queueing system may also be set up by defining a batch queue in the **/etc/qconfig** file and using the **enq** command to submit a job to this queue. This queue may be set up with a first-come, first-serve discipline. The following stanzas should be added to the **/etc/qconfig** file to enable this:

```
bsh
device = bshdev
discipline = fcfs
bshdev:
backend = usr/bin/sh
```

This configuration may already exist in the **/etc/qconfig** file. If you want your commands and programs to run under the Korn shell, you should change the last line in the above stanza to:

```
backend = usr/bin/ksh
```

After creating the above stanza in the **/etc/qconfig** file, enable the queue by issuing the following:

```
qchk -A
```

Programs and commands may now be run on a first-come, first-serve basis using the **enq** command. For example, to run the program PROGRAM1 from the bsh queue, enter:

```
enq -P bsh PROGRAM1
```

The flags for the batch facility and queueing are:

at -qa
: This is for queueing **at** jobs.

at -qb
: This is for queueing batch jobs.

at -qe
: This is for queueing **ksh** jobs.

at -qf
: This is for queueing **csh** jobs.

Examples

1. To set the **at** command job queue to handle 4 concurrent jobs with a nice value of 1 and no retries, enter:

   ```
   a.4j1n
   ```

2. To set the **crontab** command job queue to handle 2 concurrent jobs with a nice value of 2 and a retry in 90 seconds if the **fork** subroutine fails, enter:

   ```
   c.2j2n90w
   ```

Implementation Specifics

The **queuedefs** file is part of Base Operating System (BOS) Runtime.

Related Information

The **at** command, **batch** command, **crontab** command, **csh** command, **enq** command, **ksh** command, **rc** command.

The **cron** daemon.

The **fork** subroutine,

rc.net File Format for TCP/IP

Purpose

Defines host configuration for network interfaces, host name, default gateway, and 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 the **init** command after the second phase of configuration has completed and after the **init** command 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 high-level configuration commands such as the **mkdev** command. The network interface configuration information is held in the running system only and must be reset at each system restart.

```
/usr/lib/methods/defif >>
$LOGFILE   2>&1
/usr/lib/methods/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 **x25ip** command for X.25 interfaces and the **slattach** command for SLIP connections. The **x25ip** 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.

At times, when diskless clients reboot using these configuration methods they hang on LED 581. This happens because diskless clients use server disk space to store the logging information. To get the client to reboot when this happens, execute the **/usr/lib/methods/cgfig** configuration method in the client **rc.net** file that resides on the server without message logging as follows:

```
/usr/lib/methods/cfgif   $*
```

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** and **cfginet** subroutines to alter the ODM database for the `inet0` object.

```
/usr/lib/methods/definet >>
$LOGFILE 2>&1/usr/lib/methods/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:

```
/usr/sbin/ifconfig Interface inet
InternetAddress up>>$LOGFILE 2 &1
```

The *Interface* parameter should specify the type and number of the interface, for example, `tr0`. The *InternetAddress* parameter should specify the Internet address of the interface, for example, `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.

```
/usr/bin/hostname  Hostname >>
 $LOGFILE 2>&1/usr/sbin/route  add  0
 Gateway >> $LOGFILE 2>&1
/usr/sbin/route  add  DestinationAddress
Gateway >>$LOGFILE 2>&1
```

The **add** variable 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/sbin/hostid ‘hostname’
/usr/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 loads and configures the NFS kernel extension:

```
if [ -x /usr/sbin/gfsinstall -a
 -x /usr/lib/drivers/nfs.ext ] ; then
    /usr/sbin/gfsinstall -a /usr/lib/drivers/
nfs.ext >>$LOGFILE 2>&1fi
```

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:

   ```
   /usr/sbin/ifconfig tr0 inet
    192.1.8.0 up >>$LOGFILE 2>&1
   ```

   This stanza defines Token-Ring interface `tr0`, with the Internet address `192.1.8.0`.

2. To set the host name, using the **hostname** command, include the following stanza:

   ```
   /usr/bin/hostname robo.austin.century.com
      >>$LOGFILE 2>&1
   ```

   This stanza sets host name `robo.austin.century.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:

   ```
   /usr/sbin/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 the default gateway.

4. To set up a static route, using the **route** command, include the following stanza:

```
/usr/sbin/route add net
192.100.201.7 192.100.13.7>>$LOGFILE 2>&1
```

The `192.100.201.7` address is the receiving computer (the *Destination* parameter). The `192.100.13.7` address is the routing computer (the *Gateway* parameter).

Implementation Specifics

This file is part of TCP/IP Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/etc/rc.tcpip	Initializes daemons at each system restart.

Related Information

The **hostname** command, **ifconfig** command, **init** command, **mkdev** command, **route** command, **sendmail** command, **slattach** command.

The **cfgif** method, **cfginet** method, **defif** method, **definet** method.

The **rc.tcpip** file.

The **inetd** daemon.

Installation and Configuration for TCP/IP in *AIX Version 4.3 System Management Guide: Communications and Networks*.

rc.ntx File Format

Purpose

Supplies configuration information for the Network Terminal Accelerator adapter card.

Description

The **/etc/rc.ntx** file invokes the **hty_load** command to load the **/etc/hty_config** file. This file can also specify a route to a gateway, using the **ntx_route** command. Also, the **rc.ntx** file enables SNMP.

The **/etc/rc.ntx** file can be used to perform different configuration tasks. For example, to supply a route to an additional gateway, add the following line immediately after the comment about additional routes, and supply an IP address for the *Destination* and *Gateway* parameters:

```
/usr/bin/ntx_route -drhp$i net Destination
Gateway
```

Following is the file as it is shipped with the software package. You can add additional commands to the file, as indicated above.

```
echo "Executing hty_load"
/usr/bin/hty_load -f /etc/hty_config
echo "Finished executing hty_load"

#
# Maximum number of Network Terminal Accelerator adapters
# supported on each workstation.
#

MAX_RHP_DEVICES=7

i=0
while [ $i -le $MAX_RHP_DEVICES ]
do
     if [ -f /etc/rhp$i.ntx_comun.conf ]; then
           echo "Configuring SNMP communities on NTX
                     Adapter rhp$i"
           /usr/bin/ntx_comun -d /dev/rhp$i -f
                     /etc/rhp$i.ntx_comun.conf
     fi
     if [ -f /etc/rhp$i.ntx_traps.conf ]; then
             echo "Configuring SNMP traps on NTX Adapter rhp$i"
             /usr/bin/ntx_traps -d /dev/rhp$i -f
                     /etc/rhp$i.ntx_traps.conf
     fi
     if [ -f /etc/rhp$i.ntx_nms.conf ]; then
             echo "Configuring SNMP nms on NTX Adapter rhp$i"
             /usr/bin/ntx_nms -d /dev/rhp$i -f
                    /etc/rhp$i.ntx_nms.conf
     fi
     if [ -f /etc/rhp$i.ntx_descr.conf ]; then
             echo "Configuring SNMP site-specific variables on
```

```
                NTX Adapter rhp$i"
        /usr/bin/ntx_descr -d /dev/rhp$i -f
                /etc/rhp$i.ntx_descr.conf
    fi
    if [ -c /dev/rhp$i ]; then
        STATE=‘lsattr -E -l rhp$i -a snmp -F value‘
        echo "Turning $STATE SNMP on NTX Adapter rhp$i"
        /usr/bin/ntx_snmp -d /dev/rhp$i $STATE
    fi
    # Additional routes for each NTX Adapter can be added here
    # example: /usr/bin/ntx_route -d /dev/rhp$i X.X.X X.X.X.X

    i=‘expr $i + 1‘ # increment count

done
```

Related Information

The **hty_load** command, **ntx_comun** command, **ntx_descr** command, **ntx_nms** command, **ntx_route** command, **ntx_snmp** command, and **ntx_traps**

remote File Format for tip

Purpose

Describes remote systems contacted by the **tip** command.

Description

The **/etc/remote-file** file describes the remote systems that can be contacted using the **tip** command. When a user invokes the **tip** command, the command reads the **remote** file to find out how to contact the specified remote system. If invoked with the *SystemName* parameter, the **tip** command searches the **remote** file for an entry beginning with that system name. If invoked with the *PhoneNumber* parameter, the command searches the **remote** file for an entry beginning with **tip***BaudRate*, where *BaudRate* designates the baud rate to be used for the connection.

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 operating system 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 **REMOTE** environment variable before issuing the **tip** command, or use the **tip** command **remote** variable.

A sample **remote** file for **tip** is included with the operating system. The sample file is named **/usr/lib/remote-file**. This sample file contains two examples, either of which is a complete **remote** file. One of the examples uses a set of general dialer definitions, followed by general system definitions, and 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** file is a system name, baud rate, or dialer name followed by a description and one or more attributes, as follows:

SystemName[**|***SystemName* ...]**|** *Description***:***Attribute*[**:***Attribute* ...]**:**

OR

tip*BaudRate***|***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 system or dialer name can be given; in this case, they must be separated by pipe symbols and precede the *Description* parameter. The last section in this list is always treated by the **tip** command as a description, not a system name.

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 on the next line by typing a \ (backslash). The continuation line must begin with a : (colon) and can be indented for readability.

Any line beginning with a # (pound sign) is read as a comment line.

Note: Spaces can be used only within the *Description* parameter or in comment lines.

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 calling 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 is linked to 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 1200 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* parameter.

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
: Makes 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 link 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 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 setting is null value.

oe=*OutputString* — Specifies the output end-of-file mark. The default setting is a null value.

pa=*Parity* — Specifies the required parity setting for connecting to the remote system. The default setting is Even. Valid choices are: Even (7 bits, even parity), Odd (7 bits, odd parity), None (7 bits, no parity), and Graphic (8 bits, no parity).

pn= — Lists telephone numbers to be used to call the remote system. This entry is required if a modem is used to call a remote system, except in a **tip***BaudRate* entry when a telephone number is entered with the **tip** command.

If the **tip** command is invoked with the *PhoneNumber* parameter, the **pn** attribute in the appropriate **tip***BaudRate* entry is ignored and the number given when the command is invoked is used instead.

The **pn** attribute can be in either of the following forms:

pn=@ — Instructs **tip** to search the **/etc/phones-file** 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 call 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. The *DialerName* entry must be defined elsewhere in the **remote** file.

tc=*SystemName* — Includes the specified *SystemName* entry. The *SystemName* entry must be defined elsewhere in the **remote** file.

Setting Up Group Entries

Set up entries in the **remote** file in two ways. Define each system individually, giving all of its attributes in that entry. This works well if you are contacting several dissimilar systems.

Or group the systems by similarity. To do this, use two or three groups, depending on how the systems are similar. The groups can be arranged by:

- 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. Use the **tc** attribute to refer to a dialer entry.
- Specific system descriptions, which use the **tc** attribute to refer to one of the general system types or a dialer entry.

You can omit either the dialer definitions or the general system definitions, depending on the way the remote systems are grouped.

Examples

Defining a System Individually

To define a system without using the **tc=** attribute, enter:

```
vms750|ghost|NPG 750:\
   :dv=/dev/tty36,/dev/tty37:br#9600:el=^Z^U^C^S^Q^O:\
   :ie=$@:oe=^Z:
```

This entry defines system `vms750`, which can also be referred to as `ghost`. 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 and a general system entry, followed by specific system entries that refer to the general entries.

1. To define a dialer, enter:

   ```
   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` and is an ACU type of `dn11`. The dial-up (`du`) flag is set.

2. To define a general system type and refer to a dialer entry, enter:

   ```
   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.

3. To describe a specific system, enter:

   ```
   zeus|CSRG ARPA VAX-11/780:pn=@:tc=unix1200:
   ```

   This entry describes system `zeus`, which is described as a `CSRG ARPA VAX-11`. The **tip** command then searches the **/etc/phones** file for the telephone number (`pn=@`) and uses the attributes of a `unix1200` system type (`tc=unix1200`).

Implementation Specifics

This file is part of the Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/remote	Denotes the complete path name of the **remote** file.
/etc/phones	Lists the phone numbers used to contact remote systems.
/usr/lib/remote-file	Contains an example **remote** file.

Related Information

The **tip** command.

resolv.conf File Format for TCP/IP

Purpose

Defines Domain Name Protocol (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 protocol to request name resolution services from a remote DOMAIN name server host. If no **resolv.conf** file exist than the resolver routines continue searching their direct path, which may include searching through **/etc/hosts** file or the NIS hosts map.

> **Note:** If the **resolv.conf** file does not exist, the resolver routines attempt name resolution using the default paths, the **/etc/netsvc.conf** file, or the NSORDER environment variable.

If the host is a name server, the **resolv.conf** file must exist and contain a nameserver reference to itself as well as a default domain.

The **resolv.conf** file can contain one `domain` entry or one `search` entry, a maximum of three `nameserver` entries, and any number of `options` entries.

A `domain` entry tells the resolver routines which default domain name to append to names that do not end with a . (period). 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` or `search` entry in the file, the **gethostbyname** subroutine returns the default domain (that is, everything following the first period). If the host name does not have a domain name included, the root domain is assumed.

A `search` entry defines the list of domains to search when resolving a name. Only one `domain` entry or `search` entry can be used. If the `domain` entry is used, the default search list is the default domain. A `search` entry should be used when a search list other than the default is required. The entry is of the form:

```
search DomainName ...
```

The `search` entry can have from one to six *DomainName* variables. The first *DomainName* variable is interpreted as the default domain name. The `DomainName` variable is the name of a domain that should be included in the search list.

Notes:

1. The `domain` entry and `search` entry are mutually exclusive, so if both entries are used, the one that appears last will override the other.

2. The resolver routines require you to set the default domain. If the default domain is not set in the **/etc/resolv.conf** file, then you must set it in the hostname on the machine.

A `nameserver` entry defines the Internet address of a remote DOMAIN name server to the resolver routines on 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.

The *Address* variable is the address of the preferred network on which you want the address returned. The *Netmask* variable is the netmask of the corresponding network address.

The options entry specifies miscellaneous behaviors of the resolver. The entry is of the form:

```
options OptionName
```

The OptionName variable can have one of the following values:

debug Turns on the RES_DEBUG resolver option, which enables resolver debugging.

ndots:n Specifies that for a domain name with *n* or more periods (.) in it, the resolver should try to look up the domain name "as is" before applying the search list.

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.

Examples

To define a domain host that is not a name server, enter:

```
domain abc.aus.century.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 Base Operating System (BOS) Runtime.

Files

/usr/lpp/tcpip/samples/resolv.conf Contains the sample **resolv.conf** file.

Related Information

The **namerslv** command.

The **named** daemon.

The **/etc/hosts** file format.

The **gethostbyaddr** subroutine, **gethostname** subroutine.

TCP/IP Name Resolution.

Configuring Name Servers and Naming in

.rhosts File Format for TCP/IP

Purpose

Specifies remote users that can use a local user account on a network.

Description

The **$HOME/.rhosts** file defines which remote hosts (computers on a network) can invoke certain commands on the local host without supplying a password. This file is a hidden file in the local user's home directory and must be owned by the local user. Although you can set any permissions for this file, it is recommended that the permissions of the **.rhosts** file be set to 600 (read and write by owner only). The format of the **$HOME/.rhosts** file is:

```
HostNameField [UserNameField]
```

When a remote command executes, the local host uses the local **/etc/hosts.equiv** file and the **$HOME/.rhosts** file of the local user account to validate the remote host and remote user.

Host-Name Field

The **.rhosts** file supports the following host-name entries:

```
+
HostName
-HostName
+@NetGroup
-@NetGroup
```

A + (plus sign) signifies that any host on the network is trusted. The `HostName` entry is the name of a remote host and signifies that any user logging in from `HostName` is trusted. A `-HostName` entry signifies that the host is not trusted. A `+@NetGroup` or `-@NetGroup` entry signifies that all hosts in the netgroup or no hosts in the netgroup, respectively, are trusted.

The *@NetGroup* parameter is used by Network Information Service (NIS) for grouping. Refer to the NIS **netgroup** file for more information.

User-Name Field

The **.rhosts** file supports the following user-name entries:

```
+
UserName
-UserName
+@NetGroup
-@NetGroup
```

A + (plus sign) signifies that any user on the network is trusted. The *UserName* entry is the login name of the remote user and signifies that the user is trusted. If no user name is specified, the remote user name must match the local user name. A *-UserName* entry signifies that the user is not trusted. A *+@NetGroup* or *-@NetGroup* entry signifies that all users in the netgroup or no users in the netgroup, respectively, are trusted.

The *@NetGroup* parameter is used by NIS for grouping. Refer to the NIS **netgroup** file for more information.

Examples

1. To allow remote users to log in to a local-user account, enter:

   ```
   hamlet dewey
   hamlet irving
   ```

   These entries in the local user's **$HOME/.rhosts** file allow users `dewey` and `irving` at remote host `hamlet` to log in as the local user on the local host.
2. To prevent any user on a given remote host from logging in to a local-user account, enter:

   ```
   -hamlet
   ```

   This entry in the local user's **$HOME/.rhosts** file prevents any user on remote host `hamlet` from logging in as a local user on the local host.
3. To allow all hosts in a netgroup to log in to a local-user account, while restricting specified users, enter:

   ```
   +@century -joe
   +@century -mary
   +@century
   ```

   This entry in the local user's **$HOME/.rhosts** file allows all hosts in the `century` netgroup to log in to the local host. However, users `joe` and `mary` are not trusted, and therefore are requested to supply a password. The deny, or **-** (minus sign), statements must precede the accept, or + (plus sign), statements in the list. The @ (at sign) signifies the network is using NIS grouping.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

/etc/host.equiv	Specifies remote systems that can execute commands on the local system.
netgroup	Lists the groups of users on the network.

Related Information

The **lpd** command, **rcp** command, **rdist** command, **rdump** command, **rlogin** command, **rsh** command, **ruser** command.

The NIS **netgroup** file.

The **rlogind** daemon, **rshd** daemon.

The TCP/IP **hosts.equiv** file format.

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 logical parts:

Checksum	The logical sum of all characters except the characters in the first line
Delta table	Information about each delta including type, SCCS identification (SID) number, date and time of creation, and comments about the delta
User Names	Login names, group names, or numerical group IDs of users who are allowed to add or remove deltas from the SCCS file
Header flags	Flags defining how some SCCS commands work with the SCCS file, or defining values for identification keywords in the file
Comments	Descriptive information about the file
Body	The actual text lines intermixed with control lines

Note: 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 the @ (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. Text lines cannot begin with the control character.

Checksum

The checksum is the first line of an SCCS file. This line has the following format:

@h*Number*

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 logical sum of all characters in the SCCS file (not including the characters in this line). It is recalculated each time the SCCS file is updated with SCCS commands, and is used to detect possibly damaging changes made to an SCCS file by non-SCCS commands.

Delta Table

Each time a group of changes, known as a delta, is made to an SCCS file, the delta table creates a new entry. Each entry contains descriptive information about the delta. The @s (at sign, letter s) character defines the beginning of a delta table entry, and the @e (at sign, letter e) character defines the end of the entry. For each delta created, there is a delta table entry in the following format:

```
@s NumberLinesInserted/NumberLinesDeleted/NumberLinesUnchanged
@d DeltaType SID Date Time UserID Number PreNumber
@i NumbersIncluded . . .
@x NumbersExcluded . . .
@g NumbersIgnored . . .
@m ModificationRequestNumber
@c Comments . . .
```

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 from the previous delta.

@d Designates the second line of each entry, which contains the following variables:

DeltaType	Type of delta. The letter d designates a normal delta; the letter r designates a delta that has been removed with the **rmdel** command.
SID	SCCS ID (SID) of the delta.
Date	Date, in the YY/MM/DD format, that the delta was created.
Time	Time, in the HH:MM:SS format, that the delta was created.
UserID	Login name that corresponds to the real user ID at the time the delta was created.
Number	Serial numbers of the delta.
PreNumber	Serial numbers of the delta's predecessor.

@i Indicates the serial numbers of the deltas that are included in the creation of this delta by using the **get** command with the **-i** flag. This line can contain several delta numbers and is optional.

@x Indicates the serial numbers of the deltas that were excluded from the creation of this delta by using the **get** command with the **-x** flag. This line can contain several delta numbers and is optional.

@g Indicates the serial numbers of the deltas that were ignored in the creation of this delta by using the **delta** command with the **-g** flag. This line can contain several delta numbers and 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

This section of the file contains the list of login names, group names, or 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. The list is changed using the **admin** command with the **-a** or **-e** flag.

Header Flags

Flags control commands and define keywords used internally in the SCCS. Header flags are set using the **admin** command with various flags. The format of each line is:

`@f` *`Flag Text`*

The control character and variables in the header flags section have the following meanings:

@fb Branch. Allows the use of the **-b** flag of the **get** command to cause a branch in the delta tree.

@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.

@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*.

@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 causes an error and the delta fails.

@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 the **get** command using the **-e** flag.

@fm Module name. Defines the replacement of a module name for the **%M%** identification keyword. This value is used to override the default.

@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 created. For example, delta 5.1 is created after delta 2.1, skipping releases 3 and 4. When this flag is omitted, skipped releases are omitted from the delta table.

@fq User-defined flag. Defines the replacement of the **%Q%** identification keyword.

@ft Type of program. Defines the replacement of the **%Y%** identification keyword.

@fv Program name. Controls prompting for MR numbers in addition to comments on delta creation. If a value is assigned, it defines an MR number validity-checking program.

Comments

When comments are taken from a file containing descriptive text by using the **admin** command with the **-t** flag option, the contents of that file are displayed in the comments section. Typically, the comments section contains a description of the purpose of the entire file and 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 consists of two types of lines: control lines and text lines. Control lines bracket text lines. The text lines contain pieces of text that were inserted or deleted for a particular version of the file. The control lines that bracket a piece of text indicate whether a piece of text was inserted or deleted, and in what version. When a particular version of a file is created from the SCCS file, the control lines identify the pieces of text that should be added or deleted for that version of the file.

Control lines can be nested within one another, so the same portion of text can be bracketed by several sets of control lines. The body of a long SCCS file can be very complicated. The SCCS commands, however, provide a better way to understand the different versions of an SCCS file.

@I*Number* Indicates an insert control line. The *Number* variable indicates the serial number that corresponds to the delta for the control line. Text inserted between this control line and an end control line with the same serial number was inserted as part of the delta that corresponded to the same serial number.

@D*Number* Indicates a delete control line. The *Number* variable indicates the serial number that corresponds to the delta for the control line is indicated by the *Number* variable. Text deleted between this control line and an end control line with the same serial number was deleted as part of the delta that corresponded to the same serial number.

@E*Number* Indicates an end control line. The serial number that corresponds to the delta for the control line is indicated by the *Number* variable. This indicates the end of a section of text to be inserted or deleted.

Within the text are also identification keywords that are specific to the SCCS file system. These keywords represent identifying information about the SCCS file. When using the **get** command without the **-e** or **-k** flag, these keywords will be replaced by their values. Because different versions have different identifying information, the identification keywords provide an easy way for the SCCS file system to provide the correct identifying information for any version of the file requested by the **get** command. Keywords can be used to provide several kinds of information:

- Version identification information:

Keyword	Value
%M%	Module name; the value of the **m** header flag in the SCCS file
%I%	SID (**%R%**, **%L%**, **%B%**, **%S%**)
%R%	Release
%L%	Level
%B%	Branch
%S%	Sequence

- Time and date information:

Keyword	Value
%D%	Date of the current **get** command (YY/MM/DD)
%H%	Date of the current **get** command (MM/DD/YY)
%T%	Time of the current **get** command (HH:MM:SS)
%E%	Date newest applied delta was created (YY/MM/DD)
%G%	Date newest applied delta was created (MM/DD/YY)
%U%	Time *newest applied* delta was created (HH:MM:SS)

- Name information:

Keyword	Value
%F%	SCCS file name
%P%	Full path name of the SCCS file

- Flag values:

Keyword	Value
%Q%	Value of the **-q** header flag in the SCCS file
%Y%	Module type; the value of the **-t** header flag in the SCCS file

- Line numbers:

Keyword	Value
%C%	The current line number. This keyword identifies message output by the program. It should not be used on every line to provide sequence numbers.

- Constructing **what** strings:

Keyword **Value**

%A% A shorthand notation for constructing **what** strings for program files specific to other operating systems. Its value equals the following key letters:

```
%A% = %Z%%Y% %M% %I%%Z%
```

%W% A shorthand notation for constructing **what** strings for program files specific to this operating system. Its value is the characters and key letters:

```
%W% = %Z%%M%<horizontal-tab>%I%
```

%Z% The 4-character string @(#) (at sign, left parenthesis, pound sign, right parenthesis) recognized by the **what** command.

Related Information

Source Code Control System (SCCS) Overview in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs* contains general information about the SCCS file system.

services File Format for TCP/IP

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 on a single line corresponding to the form:

ServiceName PortNumber/ProtocolName Aliases

These fields contain the following information:

ServiceName	Specifies an official Internet service name.
PortNumber	Specifies the socket port number used for the service.
ProtocolName	Specifies the transport protocol used for the service.
Aliases	Specifies a list of unofficial service names.

Items on a line are separated by spaces or tabs. Comments begin with a # (pound sign) and continue until the end of the line.

If you edit the **/etc/services** file, run the **refresh -s inetd** or **kill -1** *InetdPID* command to inform the **inetd** daemon of the changes.

Examples

Entries in the **services** file for the **inetd** internal services may 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/udp      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 Base Operating System (BOS) Runtime.

Related Information

The **getservent** subroutine.

The **/etc/inetd.conf** file format.

Object Data Manager (ODM) Overview for Programmers in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*.

setmaps File Format

Purpose

Defines the text of a code-set map file and a terminal map file.

Description

The text of a code set map file consists of a description of the code set. The text of a terminal map file consists of a set of rules.

Code-Set Map File

The text of a code set map file is a description of the code set. It specifies the optional converter modules to push on the stream. The code set map file is located in the **/usr/lib/nls/csmap** directory. Its name is the code set name.

The code set map file contains the following lines:

Name :	*name*
Type :	**M | S**
Multibyte handling :	**EUC**
ioctl EUC_WSET :	*w1***:***d1***,** *w2***:***d2***,** *w3***:***d3*
lower converter :	**/usr/lib/drivers/***lwconv*
upper converter :	**/usr/lib/drivers/***upconv*

The lines have the following meaning:

Name Specifies the code set name. It is also the code set map file name.

Type Specifies the code set type. It can be one of the following:

M Denotes a multibyte code set.

S Denotes a single byte code set.

Multibyte handling Specifies the type of multibyte handling of the code set. This line is required only if **Type** is **M**. It must be **EUC**, denoting an EUC multibyte code set.

ioctl EUC_WSET Specifies the parameters for the **EUC_WSET** ioctl operation. This line is required only if **Type** is **M**. The *w1*, *w2*, and *w3* parameters specify the memory width of the code set; the *d1*, *d2*, and *d3* parameters specify the screen width of the code set.

lower converter

upper converter Specifies the lower and upper converters to use on the stream. This line is required only if the code set is a non-EUC multibyte code set.

For example, the code set map file for the ISO 8859-1 code set would contain the following lines:

```
Name:   ISO8859-1
Type:   S
```

Another example: the code set map file for the IBM-943 code set would contain the following lines:

```
Name :                  IBM-943
Type :                  M
Multibyte handling :    EUC
ioctl EUC_WSET :        2:2,1:1,2:2
lower converter :       /usr/lib/drivers/lc_sjis
upper converter :       /usr/lib/drivers/up_sjis
```

Terminal Map File

The text of a terminal map file is a set of rules. Each rule has the following format:

```
pattern:replacement
```

The size of the input pattern string is limited to 10 characters in length and the size of the replacement pattern string is limited to 16 characters in length.

The pattern string can include the following special characters:

?	Matches any single byte.
@x	Matches this rule only if the pattern processor is in state *x*, where *x* is any single byte. (This sequence does not match a character in the input buffer.)
\?, **\@**, or ****	Prevents the pattern processor from interpreting ? (question mark), @ (at sign), or \ (backslash) as special characters.
\ddd	Represents any byte in octal notation.
\xdd	Represents any byte in hexadecimal notation.

The replacement string can include the following special characters:

$n	Uses the *n*th character in the input string that matched this pattern, where *n* is a decimal digit.
@x	Moves the pattern processor into state *x*. (This sequence does not become part of the replacement string.)
\$, **\@**, or ****	Prevents the pattern processor from interpreting $, @, or \ as special characters.
\ddd	Represents any byte in octal notation.
\xdd	Represents any byte in hexadecimal notation.

Files

/usr/lib/nls/csmap/sbcs	Code set map for a single-byte code page
/usr/lib/nls/csmap/IBM-932	Code set map for the IBM-932 code page
/usr/lib/nls/csmap/IBM-943	Code set map for the IBM-943 code page
/usr/lib/nls/csmap/IBM-eucJP	Code set map for the IBM-eucJP code page
/usr/lib/nls/csmap/IBM-eucKR	Code set map for the IBM-eucKR code page
/usr/lib/nls/csmap/IBM-eucTW	Code set map for the IBM-eucTW code page
/usr/lib/nls/termmap/*.in	Input map files
/usr/lib/nls/termmap/*.out	Output map files

Related Information

The **eucioctl.h** file.

The **setmaps** command.

The **setcsmap** subroutine.

simprof File Format

Purpose

Specifies PC Simulator startup options.

Description

When you start PC Simulator with the **pcsim** command, PC Simulator searches for a profile of startup options. The profile used by PC Simulator is the **simprof** file format. It is a pure ASCII text file that you can edit with any text editor.

You can specify the name of a profile with the **-profile** flag at the **pcsim** command. If you do not enter a **-profile** flag, PC Simulator searches for the **simprof** default profile. This sample profile, included with PC Simulator, is located in the **/usr/lpp/pcsim/samples** directory.

You can define more than one profile. These profiles can be for different users or for starting PC Simulator with different options. PC Simulator first searches for the specified profile in the current working directory, then in the **$HOME** directory, and finally in the **/usr/lpp/pcsim** directory. To operate with only one profile, you can copy the **simprof** sample profile to one of these directories, and edit it to set the options you want.

Even if PC Simulator finds a profile, it searches all three directories. It can, therefore, find more than one profile with the same file name. If this happens, PC Simulator accumulates options from each profile. It overlays values for the same option in each profile and uses the last value it reads. You can set options with flags from the command line that override any options in a profile.

Examples

A simulator profile resembles an AIXwindows default profile. 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 that 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. PC Simulator expands environment variables inside the **simprof** file.

> **Note:** If there is no diskette drive present, the entries for `Adiskette` and `Bdiskette` should be removed from the profile. If there is only one diskette drive present, the entry for `Bdiskette` should be removed from the profile.

```
Cdrive        : /home/dos1/txt.fil  # select file /home/dos1/txt.fil
                                    # for fixed disk C:
Ddrive        : /home/dos2          # select directory /home/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
display       :                     # use default AIXwindows
                                    # server, unix:0
dmode         : V                   # select VGA display mode
geometry      :                     # use default window size
                                    # & position, 720x494+152+265
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       : 100                 # refresh display every
                                    # 100 milliseconds
lpt1          : lp0                 # emulate DOS lpt1 with AIX lp0
lpt2          :                     # none selected
lpt3          :                     # none selected
mouse         : com1                # emulate Microsoft serial mouse
ptime         : 30                  # print job file buffering
                                    # time out after 30 seconds
xmemory       : 1024                # provide 1MB extended memory
```

Files

/usr/lpp/pcsim/samples/simprof

Standard Resource Record Format for TCP/IP

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*} *AddressClass* *RecordType* *RecordSpecificData*

Field Definitions

Name	Varies depending on the *RecordType* 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 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.
AddressClass	Address class of the record. There are three valid entries for this field: ANY for all address classes, IN for Internet, and CHAOS for Chaos net.
RecordType	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
	Details and examples of record types are given below.
RecordSpecificData	These fields are dependent on the *RecordType* 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 . (period) indicates the current domain.

Note: Use the . (period) at the end of resource records to append the path of the current domain.

. .	If used in the *Name* field, two periods indicate the null domain name of the root domain.
@	If used in the *Name* field, an @ (at sign) indicates the current origin.
\X	Where X is any character except numbers 0 through 9 or the character **.** (period), a backslash preceding a character indicates that the character's special meaning should not be used. For example, **\@** (backslash, at sign) can be used to put an @ character in the label of an entry in the *Name* field.
\DDD	Where each D is any number between 0 and 9. Each number is identified as the binary octet corresponding to the number. These octets are not checked for special meaning.

Note: The \DDD character is not used in the *Name* field of a resource record.

( )	Parentheses indicate that data broken into more than one line should be grouped together. The () (parentheses) are currently used in the **SOA** and **WKS** resource records.
;	Indicates a comment line. All characters after the ; (semicolon) are ignored.
*	An * (asterisk) indicates wildcards.

Note: The * (asterisk) character is not used in the *Name* field of a resource record.

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 **$INCLUDE** and **$ORIGIN** lines.

$INCLUDE *FileName*
: This line begins in column one and is followed by a file name. It indicates that the specified file 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 is a list of the resource record types used in the **named** data files:

- Start of authority record
- Name server record
- Address record
- Host information record
- Well-known services record
- Canonical name record
- IN-ADDR.ARPA record
- Domain-name pointer record
- Gateway PTR record
- Mailbox record
- Mail rename name record
- Mailbox information record
- Mail group member record
- Mail exchanger record

Start of Authority Record

The start of authority (**SOA**) record indicates the start of a zone of authority. There should be only one start of authority record per zone, indicated by a value of **SOA** in the *RecordType* field. 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 corresponds to the following format:

```
{Name}{TTL} AddressClass RecordType   Origin            PersonInCharge
@            IN           SOA        merl.century.com  jane.merl.century.com
             (1.1         ;Serial
             3600         ;Refresh
             600          ;Retry
             3600000      ;Expire
             86400)       ;Minimum
```

Fields

Name	Name of the zone.
TTL	Time to live.
AddressClass	Internet (IN).
RecordType	Start of authority (**SOA**).
Origin	Name of the host on which this data file resides.
PersonInCharge	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).
`Serial`	Version number 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 can 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).

Name Server Record

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, indicated by a value of **NS** in the *RecordType* field. 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 corresponds to the following format:

{Name} {TTL} AddressClass RecordType NameServerName

```
            IN        NS       arthur.century.com
```

Fields

Name	Indicates the domain serviced by the specified name server. In this case, the domain defaults to the value in the previous resource record.
TTL	Time to live.
AddressClass	Internet (IN).
RecordType	Name server (**NS**).
NameServerName	The name server responsible for the specified domain.

Address Record

The address record specifies the address for the host and is indicated by a value of **A** in the *RecordType* field. Address records can be entries in the **named.ca**, **named.data**, and **named.rev** files. Its structure corresponds to the following format:

{Name} *{TTL}* *AddressClass* *RecordType* *Address*

```
arthur           IN           A      132.10.8.1
                 IN           A       10.0.4.1
```

Fields

Name
: Name of the host.

TTL
: Time to live.

AddressClass
: Internet (IN).

RecordType
: 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.century.com` was `fran.aus.century.com`, then the `NS` record and the required **A** record would look like:

```
aus.century.com.      IN      NS   fran.aus.century.com.
fran.aus.century.com. IN      A    192.9.201.14
```

Host Information Record

The host information (**HINFO**) record lists host specific information, and is indicated by **HINFO** in the *RecordType* field. 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 corresponds to the following format:

{*Name*} {*TTL*} *AddressClass* *RecordType* *Hardware* *OS*

Fields

Name	Name of the host.
TTL	Time to live.
AddressClass	Address class. Valid values are IN for Internet and CHAOS for Chaos net.
RecordType	Host information (**HINFO**).
Hardware	Make and model of hardware.
OS	Name of the operating system running on the host.

Well-Known Services Record

The well-known services (**WKS**) record lists the well-known services supported by a particular protocol at a specified address. This record is indicated by **WKS** in the *RecordType* field. Although AIX TCP/IP provides the record for backward compatibility, it is now obsolete.

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 corresponds to the following format:

{Name}{TTL} AddressClass RecordType Address Protocol ListOfServices

```
             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.
TTL	Time to live
AddressClass	Internet (IN)
RecordType	Well-known services (**WKS**)
Address	Internet address of the adapter in dotted decimal form
Protocol	Protocol used by the list of services at the specified address
ListOfServices	Services supported by a protocol at the specified address

Canonical Name Record

The canonical name record specifies an alias for a canonical name (**CNAME**), and is indicated by **CNAME** in the *RecordType* field. 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 corresponds to the following format:

{Aliases} {TTL} AddressClass RecordType CanonicalName

```
knight              IN           CNAME             lancelot
john                IN           CNAME             lancelot
```

Fields

Aliases	Alias by which the host is known
TTL	Time to live
AddressClass	Internet (IN)
RecordType	Canonical name (**CNAME**)
CanonicalName	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 the structure is based on names, 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 host addresses 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

The Domain-Name Pointer record allows special names to point to some other location in the domain. This record is indicated by **PTR** in the *RecordType* field. **PTR** resource records are mainly used in IN-ADDR.ARPA records to translate addresses to names.

Note: PTR records should use official host names, not aliases.

The **PTR** record is a valid entry in the **named.rev** file. Its structure corresponds to the following format:

{Aliases} *{TTL}* *AddressClass* *RecordType* *RealName*

```
7.0               IN            PTR        arthur.century.com.
```

Fields

Aliases	Specifies where this record should point in the domain. Also specifies 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.
AddressClass	Internet (IN).
RecordType	Pointer (**PTR**).
RealName	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 for each class, A, B, and C. The `gw` 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

```
2.33.12.192.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.century.com`:

```
elizabeth                 IN    MB     venus.abc.aus.century.com.
```

Mailbox Record

The mailbox (**MB**) record defines the machine where a user wants to receive mail, and is indicated by **MB** in the *RecordType* field. The **MB** record is a valid entry in the **named.data** file. Its structure corresponds to the following format:

{Aliases} *{TTL}* *AddressClass* *RecordType* *Machine*

```
jane            IN             MB          merlin.century.com
```

Fields

Aliases	The user login ID
TTL	Time to live
AddressClass	Internet (IN)
RecordType	Mailbox (**MB**)
Machine	Name of the machine at which the user wants to receive mail

Mail Rename Name Record

The mail rename (**MR**) name record allows a user to receive mail addressed to a list of aliases. This record is indicated by **MR** in the *RecordType* field. The **MR** record is a valid entry in the **named.data** file. Its structure corresponds to the following format:

{Aliases} *{TTL}* *AddressClass* *RecordType* *CorrespondingMB*

```
merlin              IN            MR                   jane
```

Fields

Aliases	Alias for the mailbox name listed in the last field.
TTL	Time to live.
AddressClass	Internet (IN).
RecordType	Mail rename (**MR**).
CorrespondingMB	The name of the mailbox. This record should have a corresponding **MB** record.

Mailbox Information Record

The mailbox information (**MINFO**) record creates a mail group for a mailing list, and is indicated by **MINFO** in the *RecordType* field. 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 corresponds to the following format:

{Name}	*{TTL}*	*AddressClass*	*RecordType*	*Requests*	*Maintainer*
`postmaster`		`IN`	`MINFO`	`post-request`	`greg.century.com`

Fields

Name	The name of the mailbox.
TTL	Time to live.
AddressClass	Internet (IN).
RecordType	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

The mail group member (**MG**) record lists the members of a mail group. This record is indicated by **MG** in the *RecordType* field. The **MG** record is a valid entry in the **named.data** file. Its structure corresponds to the following format:

{MailGroupName}	*{TTL}*	*AddressClass*	*RecordType*	*MemberName*
`dept`		`IN`	`MG`	`Tom`

Fields

MailGroupName	Name of the mail group.
TTL	Time to live.
AddressClass	Internet (IN).
RecordType	Mail group member record (**MG**).
MemberName	The login ID of the group member.

Mail Exchanger Record

The mail exchanger (**MX**) records identify machines (gateways) that know how to deliver mail to a machine that is not directly connected to the network. This record is indicated by **MX** in the *RecordType* field. Wildcard names containing an * (asterisk) can be used for mail routing with **MX** records. There may be servers on the network that 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 corresponds to the following format:

{Name} *{TTL}* *AddressClass* *RecordType* *PrefValue* *MailExchanger*

```
Ann.bus.com       IN            MX          0      Hamlet.Century.Com
*.dev.bus.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.bus.com` should be routed through the mail gateway `Lear.Century.Com`.
TTL	Time to live.
AddressClass	Internet (IN).
RecordType	Mail Exchanger **(MX**).
PrefValue	Indicates the order the mailer should follow when there is more than one way to deliver mail to a host.
MailerExchanger	The full name of the mail gateway. See RFC 974 for more information.

Examples

The following is an example of a mailing list:

```
dept          IN       MINFO     dept-request jane.merlin.century.com
              IN       MG        greg.arthur.century.com
              IN       MG        tom.lancelot.century.com
              IN       MG        gary.guinevere.century.com
              IN       MG        kent.gawain.century.com
```

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Related Information

The **named** daemon.

The **DOMAIN Data** file format, **DOMAIN Cache** file format, **DOMAIN Local** file format, **DOMAIN Reverse Data** file format.

Naming in *AIX Version 4.3 System Management Guide: Communications and Networks*.

Sysfiles File Format for BNU

Purpose

Gives system administrators flexibility in configuring their **Systems**, **Devices** and **Dialers** files for use with BNU commands.

Description

The **/etc/uucp/Sysfiles** file let system administrators specify alternate **Systems**, **Devices** and **Dialers** files to replace the default files in the **/etc/uucp** directory or to supplement those files to enable a separation of the data needed to access remote systems. It is organized so a user can invoke two distinct types of services, **uucico** and **cu**. The **uucico** service refers to the **/usr/sbin/uucp/uucico** command and the commands that invoke it, for example **uucp**, **uux**, **uusend**, **uucico.** It automatically logs into remote systems and sends and receives data. The **cu** service connects to remote systems without attempting to login and uses the **cu**, **ct**, and **slattach** commands to contact remote systems. The responses to the user name and password prompts as well as any data transfer is the responsibility of the user. Based upon these differences a system administrator can split the data used to contact remote systems according the service types

The **Sysfiles** file contains a description of each BNU service on the local system that can establish a remote connection. Each line in the **Sysfiles** file corresponds to the following syntax:

```
service=uucico|cu [systems=filename[:filename]] \
                  [devices=filename[:filename]] \
                  [dialers=filename[:filename]] \
```

If a service does not have a corresponding line in the **Sysfiles** file, the default files are used.

Examples

1. A **Sysfiles** configuration that splits the configuration files for **uucico** and **cu** into different sets of files would be as follows:

   ```
   service=uucico  systems=Systems.cico devices=Devices.cico \
                   dialers=Dialers.cico
   service=cu      systems=Systems.cu   devices=Devices.cu    \
                   dialers=Dialers.cu
   ```

   These two lines in a **Sysfiles** file state that two separate sets of **Systems**, **Devices** and **Dialers** files are used for each service. Each service is specified by the **service=** at the beginning of a line with no leading white space. The files used for each service is named on the same line according to the substrings appended to the **systems=**, **devices=** and **dialers=**. Their default location is in the **/etc/uucp** directory.

2. A configuration to split the **uucico** and **cu** service entries into separate files, but to combine common configuration data would be as follows:

```
service=uucico  systems=Systems.cico:Systems \
                devices=Devices.cico:Devices \
                dialers=Dialers.cico:Dialers
service=cu      systems=Systems.cu:Systems  \
                devices=Devices.cu:Devices \
                dialers=Dialers.cu:Dialers
```

This example provides separate **Systems, Devices,** and **Dialers** files for each service, but combines any common data into the default files. As the example shows, multiple **Systems**, **Devices** and **Dialers** files can be specified for each service. A colon is used as the filename delimiter in such a case.

3. This example specifies separate **Systems** files for each service. Each service uses the default **Devices** and **Dialers** files.

```
service=uucico  systems=Systems.cico
service=cu      systems=Systems.cu
```

If no **Sysfiles** service entry is made for a **Systems, Devices**, or **Dialers** file, the default file is used. Any files specified in **Sysfiles** to serve as **Systems**, **Devices**, or **Dialers** files need to conform to the syntax used in the default files, **/etc/uucp/Systems**, **/etc/uucp/Devices** or **/etc/uucp/Dialers**.

Files

/etc/uucp	Contains all the default configuration files for BNU, including the **Sysfiles** file.
/etc/uucp/Sysfiles	Contains information about alternate **Systems**, **Devices** and **Dialers** files.
/etc/uucp/Systems	Lists and describes remote systems accessible to a local system, using the Basic Networking Utilities (BNU).
/etc/uucp/Devices	Contains information about available devices.
/etc/uucp/Dialers	Contains dialing sequences for various types of modems and other types of dialers.

Related Information

The **uucico** daemon, **ct** command, **cu** command, **uucp** command, **uux** command, **uusend** command

Systems File Format for BNU

Purpose

Lists and describes remote systems accessible to a local system, using the Basic Networking Utilities (BNU).

Description

BNU Systems files, **/etc/uucp/Systems** by default, list the remote computers with which users of a local system can communicate using the Basic Networking Utilities (BNU) program. Other files specified in the **/etc/uucp/Sysfiles** file can be configured and BNU Systems files. Each entry in a **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 a **Systems** file contains:

- Name of the remote system
- Times when users can connect to the remote system
- Type of link (direct line or modem link)
- Speed of transmission over the link
- Information needed to log in to the remote system

Notes:
1. When a remote system not listed in a **Systems** file attempts to contact the remote system, the BNU program calls the **/usr/sbin/uucp/remote.unknown** shell procedure.
2. Only someone with root user authority can edit a **Systems** file, which is owned by the **uucp** program login ID.

Fields in a Systems File

Each entry in a **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*

There must be an entry in every field of a line in a **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 **-** (minus sign) as a placeholder.

Lines in a **Systems** file cannot wrap. In addition, each entry must be on only one line in the file. However, a **Systems** file can contain blank lines and comment lines. Comment lines begin with a # (pound sign). Blank lines are ignored.

System Name

The *SystemName* field contains the name of the remote system. You can list an individual remote system in a **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

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`.

Multiple *Time* fields are separated by 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.

Notes:

1. This subfield, when present, overrides the default retry time of 5 minutes.
2. 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

The *Type* field identifies 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 either the **/etc/uucp/Devices** file or the **Devices** file specified in the **/etc/uucp/Sysfiles** file.

Conversation Protocol 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 a `,` (comma) and the letter representing one of the other conversation protocols, either **t** or **e**. 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 operating system 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 cannot be used when the *Type* field is `ACU` or when a modem connection is being used.
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 operating system version of the BNU program. Use the **e** protocol for a site running a nonoperating-system version of the BNU program. Use the **t** protocol for sites running the Berkeley version of the UNIX-to-UNIX Copy Program (UUCP).

Class

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 a **Systems** file is matched against the *Class* field in a **Devices** file to find the device to use for connections. For example, if a **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 **-** (minus sign) as a placeholder.

Phone

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 **-** (minus sign) 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:

1. Outside line code
2. Long-distance access codes
3. Number 1 (one) plus the area code (if the modem is out of the local area)
4. Three-digit exchange number
5. Four-digit modem number

Entering a complete 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 **/etc/uucp/Dialcodes** file to set up dial-code abbreviations.

For example, if your site communicates regularly using modems to other systems at the same remote site, it is more efficient to use a dial-code abbreviation in a **Systems** file than to type the complete phone number of each remote modem.

The dial-code entry in the **/etc/uucp/Dialcodes** file defines an alphabetic abbreviation that represents the following portions of the phone number:

- Outside line code
- Long-distance access code
- Number 1 (one) plus the area code (if the modem is out of the local area)
- Three-digit exchange number

In the *Phone* field in a **Systems** file entry, type the alphabetic abbreviation followed by the four-digit modem number.

> **Note:** Enter the alphabetic abbreviation in the **/etc/uucp/Dialcodes** file *only once* for all the remote modems listed in a **Systems** file. Then use the same abbreviation for all entries in a **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 **/etc/uucp/Dialcodes** file.

Login

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 as:

```
[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 remote system login ID. 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) the first *Expect* string, it sends its own string (the *Send* string within 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` (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 .25 to .50 seconds.
`\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 the modem 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 multispeed 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 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 a **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 `-` (minus sign) 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 multispeed 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

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

In order to make the appropriate entries in a **Systems** file, decide on the appropriate TCP/IP conversation protocol to enter in the TCP *Caller* subfield. For example, enter the following in a **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

To use a dialcode abbreviation defined in the **/etc/uucp/Dialcodes** file, enter the following in a **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

For a direct connection between two systems, a **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
```

A **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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/etc/uucp directory	Contains all the configuration files for BNU, including a **Systems** file.
/etc/uucp/Sysfiles file	Specifies possible alternative foles for the **/etc/uucp/Systems** file.
/etc/uucp/Devices file	Contains information about available devices.
/etc/uucp/Dialcodes file	Contains dialing code abbreviations.
/etc/uucp/Permissions file	Describes access permissions for remote systems.
/usr/sbin/uucp/remote.unknown file	Records contacts from unknown systems.

Related Information

The **mail** command, **sendmail** command, **uucp** command, **uucpadm** command, **uuname** command, **uuto** command, **uutry** command, **Uutry** command, **uukick** command, **uux** command.

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

telnet.conf File Format for TCP/IP

Purpose

Translates a client's terminal-type strings into **terminfo** file entries.

Description

The **telnetd** daemon uses the **/etc/telnet.conf** file during terminal negotiation to translate a client's terminal-type strings into **terminfo** file entries. The **telnet.conf** file is used when a client's terminal does not correspond directly to a **terminfo** file entry. If this is the case, the **telnet.conf** file can map standard terminal names (defined in RFC-1060 Assigned Numbers) to **terminfo** file entries that the system can emulate.

Each line in the **telnet.conf** file can contain up to 255 characters. Lines beginning with a # (pound sign) are comment lines.

The **telnet.conf** file is structured in a two-column line format, with dashes separating the items in each column. The first column specifies a manufacturer, model type, and optional additional information. The second column specifies the **terminfo** file entry that corresponds to the manufacturer, model, and optional information in the first column. The items in the first column can be either uppercase or lowercase. The items in the second column must be lowercase. RFC-1060 specifies the first terminal type in the **telnet.conf** file. The format for the **telnet.conf** file is:

```
Manufacturer-Model-Options TerminfoModel-Options
```

Security

Suggested permissions for the **telnet.conf** file are `rw-rw-r--` or `664`. Suggested ownership is `root` for owner and `system` for group.

Examples

Sample **telnet.conf** entries might look like the following:

```
DEC-VT100-AM vt100-am
diablo-1620-m8 1620-m8
h-19-a 19-a
TI-800 ti-800
```

In the first entry, the manufacturer is `DEC` (Digital Equipment Corporation), the model is `VT100`, and the `AM` option specifies automargin. In the second entry, the manufacturer is `diablo`, the model is `1620`, and the `m8` option specifies a left margin of 8 columns. In the third entry, the manufacturer is `h` (Heath), the model is `19`, and the `a` option specifies ANSII mode. In the fourth entry, the manufacturer is `TI` (Texas Instruments), and the model is `800`; no options are specified. For additional **terminfo** options, refer to the ***.ti** files in the **/usr/lib/terminfo** directory.

Implementation Specifics

This file is part of TCP/IP in Network Support Facilities in Base Operating System (BOS) Runtime.

Files

terminfo	Describes terminal by capability.

Related Information

The **telnet** command.

terminfo Directory

Purpose

Contains compiled **terminfo** source files.

Description

Terminfo is a compiled database describing the capabilities of terminals. Terminals are described in the **terminfo** source files via entries. Each entry contains information about the capabilities for a particular terminal or set of common terminals. Capabilities include the operations that can be performed, the padding requirements, cursor positioning, command sequences, and initialization sequences.

The compiled **terminfo** database is used by applications such as curses and vi that must have knowledge of the terminal but do not want to be terminal-dependent.

This article describes the **terminfo** source file format and covers the following topics:

- Source File Entries
- Types of Capabilities
- Preparing Descriptions
- Basic Capabilities
- Parameterized Strings
- Cursor Motions
- Area Clears
- Scrolling
- Insert or Delete Character
- Highlighting, Underlining, and Visual Bells
- Keypad
- Tabs and Initialization
- Miscellaneous Strings
- Status Lines
- Line Graphics
- Color Manipulation
- Special Cases
- Similar Terminals
- Printer Capabilities
- Database File Names

An example of a **terminfo** source file is provided .

This article explains the **terminfo** source file format. Before a **terminfo** description can be used by applications, the **terminfo** source file it resides in must be compiled using the **tic** command. Using the **tic** command results in the creation of one or more binaries, one for each terminal. The collection of **terminfo** binaries in a directory (usually **/usr/share/lib/terminfo**) is known as the **terminfo** database, or **terminfo**.

Source File Entries

You can edit or modify source files. A source file can contain one or more terminal descriptions or entries. A **terminfo** source file has a **.ti** suffix. Examples of source files are the **/usr/share/lib/terminfo/ibm.ti** file, which describes IBM terminals, and the **/usr/share/lib/terminfo/dec.ti** file, which describes DEC terminals.

See the **infocmp** command for obtaining the source description for a terminal when only the binary is available.

Each entry in a **terminfo** source file consists of a number of fields separated by commas. White space between commas is ignored. The following example shows a source file entry:

```
ibm6155-113|IBM 6155 Black & White display,
        font0=\E[10m,   font1=\E[11m,   font2=\E[12m,
        bold=\E[12m,    sgr0=\E[0;10m,
        cols#113,       lines#38,
        sgr=\E[%?%p1%t;7%;%?%p2%t;4%;%?%p3%t;7%;%?%p4%t;5%;%?%p6%t;12%;m,
        blink@,         use=ibm5151,
```

Entries can continue onto multiple lines by placing white space at the beginning of each subsequent line. To create a comment line, begin the line with a # (pound sign) character. To comment out an individual terminal capability, put a period before the capability name.

The first field (or line) for each terminal gives the various names by which the terminal is known, separated by | (pipe symbol) characters. The first name given should be the most common abbreviation for the terminal. (This name is the one most commonly used when setting the TERM environment variable.) The last name given should be a long name fully identifying the terminal. All other names are understood as synonyms for the terminal name. All names but the last should contain no blanks. The last name may contain blanks for readability. All names should be unique.

The remaining fields identify the terminal ’s capabilities.

When choosing terminal names, there are some conventions you should follow. The root name should represent the particular hardware class of the terminal. Do not use hyphens in the root name, except to avoid synonyms that conflict with other names. To indicate possible modes for the hardware or user preferences, append a **-** (minus sign) and one of the following suffixes:

Root Name Suffixes		
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**
-s	Status line simulation. The terminal allows for one or more lines that are normally part of the screen to be used for the status line. This is not the same as terminals that have permanently dedicated status lines.	*Terminal*-**s**
-unk	Unknown mode. This entry can be used to define a general description of a terminal that has several of the modes described above. The other entries would use the unknown entry as a base description and add the appropriate customization. See the use= field.	*Terminal*-**unk**

A terminal in 132-column mode would be *Terminal*-**w**.

Types of Capabilities

A **terminfo** entry can define any number of capabilities. All capabilities belong to one of three types:

Boolean
: Indicates that the terminal has a particular feature. Boolean capabilities are true if the corresponding name is contained in the terminal description.

Numeric
: Gives the size of the terminal or the size of particular delays.

String
: Gives a sequence that can be used to perform particular terminal operations.

This article provides tables that document the capability types. All the tables list the following:

Variable
: The name the application uses to access a capability.

Cap Name
: The short capability name. This name is used in the **terminfo** database text and by the person creating or editing a source file entry. You can use the **tput** command to output the value of a capability for a particular terminal.

I.Code
: The 2-letter internal code used in the compiled database. This code always corresponds to a **termcap** capability name.

Description
: A description of the capability.

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.

For a detailed description of the various capabilities according to function, read:

- Basic Capabilities
- Parameterized Strings
- Cursor Motions
- Area Clears
- Insert or Delete Line
- Insert or Delete Line Character
- Highlighting, Underlining, and Visual Bells
- Keypad
- Tabs and Initialization
- Miscellaneous Strings

Boolean Capabilities

A Boolean capability indicates that the terminal has some particular feature. For instance, the **am** capability in a terminal description indicates that the terminal has automatic margins (such as an automatic new line when the end of a line is reached). The following are the Boolean capabilities:

Boolean Capabilities			
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.
back_color_erase	bce	ut	Erases screen with current background.
can_change	ccc	cc	Can redefine existing color.
ceol_standout_glitch	xhp	xs	Indicates that standout is not erased by overwriting.
col_addr_glitch	xhpa	YA	Indicates only positive motion for hpa/mhpa caps.
cpi_changes_res	cpix	YF	Indicates resolution changed when changing character pitch.
cr_cancels_micro_mode	crxm	YB	Indicates cr turns off micro mode.
dest_tabs_magic_smso (or teleray_glitch)	xt	xt	Indicates destructive tabs and blanks inserted while entering standout mode.
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 or switch.
hard_copy	hc	hc	Indicates hardcopy terminal.
hard_cursor	chts	HC	Indicates cursor is hard to see.
has_meta_key	km	km	Indicates terminal has a meta key, such as shift or sets parity bit.
has_print_wheel	daisy	YC	Indicates operator needed to change character set.
has_status_line	hs	hs	Indicates terminal has a dedicated status line.
hue_lightness_saturation	hls	hl	Uses HLS color notation (Tektronix).
insert_null_glitch	in	in	Indicates insert mode distinguishes nulls.
lpi_changes_res	lpix	YG	Indicates resolution changed when changing line pitch.
memory_above	da	da	Display retained above the screen (usually multi-page terminals).
memory_below	db	db	Display retained below the screen (usually multi-page terminals)
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.
needs_xon_xoff	nxon	nx	Indicates padding will not work, that xon/xoff is required.
no_esc_ctlc (or beehive_glitch)	xsb	xb	Indicates a terminal with F1=escape and F2=Ctrl-C.
no_pad_char	npc	NP	Indicates pad character does not exist.
non_dest_scroll_region	ndscr	ND	Indicates non-destructive scrolling region.
non_rev_rmcup	nrrmc	NR	Indicates smcup does not reverse rmcup.
over_strike	os	os	Indicates terminal overstrikes.
prtr_silent	mc5i	5i	Indicates printer will not echo on screen.
row_addr_glitch	xvpa	YD	Indicates only positive motion for vpa/mvpa caps.
semi_auto_right_margin	sam	YE	Indicates printing in last column causes carriage return.
status_line_esc_ok	eslok	es	Indicates escape can be used on the status line.

tilde_glitch	hz	hz	Indicates terminal cannot print the ~ (tilde) character.
transparent_underline	ul	ul	Overstrikes with underline character.
xon_xoff	xon	xo	Indicates terminal uses xon/xoff handshaking.

Numeric Capabilities

Numeric capabilities are followed by the # (pound sign) character and a numeric value. The **cols#80** capability indicates the terminal has 80 columns. The following are the numeric capabilities:

Numeric Capabilities			
Variable	**Cap Name**	**I.Code**	**Description**
buffer_capacity	bufsz	Ya	Specifies the number of bytes buffered before printing.
columns	cols	co	Specifies the number of columns in a line.
dot_horz_spacing	spinh	Yc	Identifies the horizontal spacing of dots in dots per inch.
dot_vert_spacing	spinv	Yb	Specifies vertical spacing of pins in pins per inch.
init_tabs	it	it	Provides initial tabs every specified number of spaces.
label_height	lh	lh	Specifies the number of rows in each label.
label_width	lw	lw	Specifies the number of columns in each label.
lines	lines	li	Specifies the number of lines on screen or page.
lines_of_memory	lm	lm	Specifies the number of lines of memory if > lines. A value of 0 indicates a variable number.
magic_cookie_glitch	xmc	sg	Indicates number of blank characters left by **smso** or **rmso**.
max_attributes	ma	ma	Identifies the maximum combined video attributes the terminal can display.
max_colors	colors	Co	Specifies the maximum number of colors supported.
max_micro_address	maddr	Yd	Indicate the limit on use of **mhpa** and **mvpa**.
max_micro_jump	mjump	Ye	Specifies the limit on use of the **mcub1**, **mcuf1**, **mcuu1**, and **mcud1** capabilities.
max_pairs	pairs	pa	Specifies the maximum number of color pairs supported.
maximum_windows	wnum	MW	Specifies the maximum number of defineable windows.
micro_char_size	mcs	Yf	Specifies the character step size when in micro mode.

micro_line_size	mls	Yg	Identifies the line step size when in micro mode.
no_color_video	ncv	NC	Indicates video attributes that cannot be used with colors.
num_labels	nlab	Nl	Specifies the number of labels on the screen. This value starts at 1.
number_of_pins	npins	Yh	Identifies the number of pins in the print-head.
output_res_char	orc	Yi	Specifies the horizontal resolution in units per character.
output_res_horz_inch	orhi	Yk	Specifies the horizontal resolution in units per inch.
output_res_line	orl	Yj	Specifies the vertical resolution in units per line.
output_res_vert_inch	orvi	Yl	Indicates vertical resolution in units per inch.
padding_baud_rate	pb	pb	Indicates lowest baud rate where carriage-return and line-return padding is needed.
print_rate	cps	Ym	Indicates print rate in characters per second.
virtual_terminal	vt	vt	Indicates virtual terminal number.
wide_char_size	widcs	Yn	Identifies the character step size when the terminal is in double-wide mode.
width_status_lines	wsl	ws	Specifies the number of columns in status lines.

String Capabilities

You define string-valued capabilities, such as the **el** capability (clear to end of line) with a 2-character code, an = (equal sign), and a string ending with a , (comma). A delay in milliseconds can appear anywhere in a string capability. To define a delay, enclose the delay between a $< and a >. The following shows the **el** capability with a delay of 3:

```
el=\EK$<3>
```

The **tputs** subroutine provides padding characters for a delay. A delay can be a number, such as `20`, or a number followed by an ***** (asterisk), such as `3*`. An asterisk indicates that the required padding is proportional to the number of lines affected by the operation. The number given represents the required padding for each affected unit. (For insert character, the factor is the number of lines affected, which is always 1, unless the terminal has the **xenl** capability and the software supports it). If you specify an asterisk, it is sometimes useful to give a delay of the form ***a.b***, such as `3.5`, to specify a delay for each unit to tenths of milliseconds. You can only specify one decimal place.

The **terminfo** database provides several escape sequences in the string-valued capabilities for easy encoding of characters. The following escape codes are recognized:

Escape Code	**Meaning**
\E,\e	Escape
\n	New line
\l	Line feed
\r	Carriage return
\t	Tab
\b	Backspace
\f	Form feed
\s	Space
\^	Caret
\\	Backslash
\,	Comma
\:	Colon
nnn	Character with octal value *nnn*
^*x*	Ctrl-*x* for any appropriate *x*
\0	Null character. `\0` actually produces \200, which does not end a string but behaves as a null character on most terminals.

The following conventions are used in the Description column of the String Capabilities table:

(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.

color_fg_6	colf6	c6	Foreground color 6, cyan.
color_fg_7	colf7	c7	Foreground color 7, white.
column_address	hpa	ch	Sets cursor column. (G)
command_character	cmdch	CC	Indicates that a terminal command-prototype character can be set.
create_window	cwin	CW	Define win #1 to go from #2, #3 to #4, #5.
cursor_address	cup	cm	Indicates screen-relative cursor motion row #1, col #2. (G)
cursor_down	cud1	do	Moves cursor down one line.
cursor_home	home	ho	Moves cursor to home position (if no **cup** addressing).
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. (G)
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** addressing).
cursor_up	cuu1	up	Moves cursor up one line.
cursor_visible	cvvis	vs	Makes cursor very visible.
define_char	defc	ZE	Define a character in a character set.
delete_character	dch1	dc	Deletes character. (*)
delete_line	dl1	dl	Deletes line. (*)
dial_phone	dial	DI	Dial phone number #1.
dis_status_line	dsl	ds	Disables status line.
display_clock	dclk	DK	Display time-of-day clock.
down_half_line	hd	hd	Indicates subscript (forward 1/2 line feed).
ena_acs	enacs	eA	Enable alternate character set.
enter_alt_charset_mode	smacs	as	Starts alternate character set.
enter_am_mode	smam	SA	Turn on automatic margins.
enter_blink_mode	blink	mb	Enables blinking.
enter_bold_mode	bold	md	Enables bold (extra bright) mode.
enter_bottom_mode	birep	bm	Starts bottom line mode. This string capability is an aid for drawing tables and is valid only for **aixterm** and **aixterm-m** terminal definitions.
enter_ca_mode	smcup	ti	Begins programs that use **cup** addressing.
enter_delete_mode	smdc	dm	Starts delete mode.
enter_dim_mode	dim	mh	Enables half-bright mode.
enter_doublewide_mode	swidm	ZF	Enable double-wide printing.
enter_draft_quality	sdrfq	ZG	Set draft quality print.
enter_insert_mode	smir	im	Starts insert mode.
enter_italics_mode	sitm	ZH	Enable italics.
enter_leftward_mode	slm	ZI	Enable leftward carriage motion.
enter_lvert_mode	lvert	lv	Starts left vertical line mode. This string capability is an aid for drawing tables. Valid only for **aixterm** and **aixterm-m** terminal definitions.
enter_micro_mode	smicm	ZJ	Enable micro motion capabilities.
enter_near_letter_quality	snlq	ZK	Set near-letter quality print.
enter_normal_quality	snrmq	ZL	Set normal quality print.
enter_protected_mode	prot	mp	Enables protected mode.
enter_reverse_mode	rev	mr	Enables reverse video mode.
enter_rvert_mode	rvert	rv	Starts right vertical line mode. This string capability is an aid for drawing tables and is valid only for **aixterm** and **aixterm-m** terminal definitions.
enter_secure_mode	invis	mk	Enables blank mode (characters are invisible).
enter_shadow_mode	sshm	ZM	Enable shadow printing.
enter_standout_mode	smso	so	Begins standout mode.
enter_subscript_mode	ssubm	ZN	Enable subscript printing.
enter_superscript_mode	ssupm	ZO	Enable superscript printing.
enter_topline_mode	topl	tp	Starts top line mode. This string capability is an aid for drawing tables and is valid only for **aixterm** and **aixterm-m** terminal definitions.
enter_underline_mode	smul	us	Starts underscore mode.
enter_upward_mode	sum	ZP	Enable upward carriage motion.
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking.
erase_chars	ech	ec	Erases #1 characters. (G)
exit_alt_charset_mode	rmacs	ae	Ends alternate character set.
exit_am_mode	rmam	RA	Turn off automatic margins.
exit_attribute_mode	sgr0	me	Disables all attributes.
exit_ca_mode	rmcup	te	Ends programs that use **cup** addressing.
exit_delete_mode	rmdc	ed	Ends delete mode.
exit_doublewide_mode	rwidm	ZQ	Disable double-wide printing.
exit_insert_mode	rmir	ei	Ends insert mode.
exit_italics_mode	ritm	ZR	Disable italics.
exit_leftward_mode	rlm	ZS	Enable rightward (normal) carriage motion.
exit_micro_mode	rmicm	ZT	Disable micro motion capabilities.
exit_shadow_mode	rshm	ZU	Disable shadow printing.
exit_standout_mode	rmso	se	Ends standout mode.
exit_subscript_mode	rsubm	ZV	Disable subscript printing.
exit_superscript_mode	rsupm	ZW	Disable superscript printing.
exit_underline_mode	rmul	ue	Ends underscore mode.
exit_upward_mode	rum	ZX	Enable downward (normal) carriage motion.
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking.
flash_screen	flash	vb	Indicates visual bell (may not move cursor).
fixed_pause	pause	PA	Pause for 2-3 seconds.
flash_hook	hook	fh	Flash the switch hook.
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). (*)
from_status_line	fsl	fs	Returns from status line.
goto_window	wingo	WG	Go to window #1.
hangup	hup	HU	Hang-up phone.
init_1string	is1	i1	Initializes terminal.
init_2string	is2	is	Initializes terminal.
init_3string	is3	i3	Initializes terminal.
init_file	if	if	Identifies file containing is long initialization strings.
init_prog	iprog	iP	Locates the program for initialization.
initialize_color	initc	Ic	Initialize the color definition.
initialize_pair	initp	Ip	Initialize color-pair.
insert_character	ich1	ic	Inserts character.
insert_line	il1	al	Adds new blank line. (*)
insert_padding	ip	ip	Inserts pad after character inserted. (*)
key_a1	ka1	K1	Specifies upper left of keypad.
key_a3	ka3	K3	Specifies upper right of keypad.
key_action	kact	kJ	Sent by action key.
key_b2	kb2	K2	Specifies center of keypad.
key_backspace	kbs	kb	Sent by backspace key.
key_beg	kbeg	@1	Beginning key. KEY_BEG
key_btab	kcbt	kB	Sent by backtab key. KEY_BTAB
key_c1	kc1	K4	Specifies lower left of keypad.
key_c3	kc3	K5	Specifies lower right of keypad.
key_cancel	kcan	@2	Cancel key. KEY_CANCEL
key_catab	ktbc	ka	Sent by clear-all-tabs key.
key_clear	kclr	kC	Sent by clear-screen or erase key.
key_close	kclo	@3	Close key. KEY_CLOSE
key_command	kcmd	@4	Command-request key.
key_command_pane	kcpn	@7	Command-pane key.
key_copy	kcpy	@5	Copy key. KEY_COPY
key_create	kcrt	@6	Create key. KEY_CREATE
key_ctab	kctab	kt	Sent by clear-tab 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	@7	End key. KEY_END
key_enter	kent	@8	Enter/send (unreliable). KEY_ENTER
key_eol	kel	kE	Sent by clear-to-end-of-line key.
key_eos	ked	kS	Sent by clear-to-end-of-screen key.
key_exit	kext	@9	Exit key. KEY_EXIT
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	k;	Sent by function key F10.
key_f11	kf11	F1	Sent by function key F11.
key_f12	kf12	F2	Sent by function key F12.
key_f13	kf13	F3	Sent by function key F13. KEY_F(13)
key_f14	kf14	F4	Sent by function key F14. KEY_F(14)
key_f15	kf15	F5	Sent by function key F15. KEY_F(15)
key_f16	kf16	F6	Sent by function key 16. KEY_F(16)
key_f17	kf17	F7	Sent by function key 17. KEY_F(17)
key_f18	kf18	F8	Sent by function key 18. KEY_F(18)
key_f19	kf19	F9	Sent by function key 19. KEY_F(19)
key_f20	kf20	FA	Sent by function key 20. KEY_F(20)
key_f21	kf21	FB	Sent by function key 21. KEY_F(21)
key_f22	kf22	FC	Sent by function key 22. KEY_F(22)
key_f23	kf23	FD	Sent by function key 23. KEY_F(23)
key_f24	kf24	FE	Sent by function key 24. KEY_F(24)
key_f25	kf25	FF	Sent by function key 25. KEY_F(25)
key_f26	kf26	FG	Sent by function key 26. KEY_F(26)
key_f27	kf27	FH	Sent by function key 27. KEY_F(27). [illegible]
key_f28	kf28	FI	Sent by function key 28. KEY_F(28)
key_f29	kf29	FJ	Sent by function key 29. KEY_F(29)
key_f30	kf30	FK	Sent by function key 30. KEY_F(30)
key_f31	kf31	FL	Sent by function key 31. KEY_F(31)
key_f32	kf32	FM	Sent by function key 32. KEY_F(32)
key_f33	kf33	FN	Sent by function key 33. KEY_F(33)
key_f34	kf34	FO	Sent by function key 34. KEY_F(34)
key_f35	kf35	FP	Sent by function key 35. KEY_F(35)
key_f36	kf36	FQ	Sent by function key 36. KEY_F(36)
key_f37	kf37	FR	Sent by function key 37. KEY_F(37)
key_f38	kf38	FS	Sent by function key 38. KEY_F(38)
key_f39	kf39	FT	Sent by function key 39. KEY_F(39)
key_f40	kf40	FU	Sent by function key 40. KEY_F(40)
key_f41	kf41	FV	Sent by function key 41. KEY_F(41)
key_f42	kf42	FW	Sent by function key 42. KEY_F(42)
key_f43	kf43	FX	Sent by function key 43. KEY_F(43)
key_f44	kf44	FY	Sent by function key 44. KEY_F(44)
key_f45	kf45	FZ	Sent by function key 45. KEY_F(45)
key_f46	kf46	Fa	Sent by function key 46. KEY_F(46)
key_f47	kf47	Fb	Sent by function key 47. KEY_F(47)
key_f48	kf48	Fc	Sent by function key 48. KEY_F(48)
key_f49	kf49	Fd	Sent by function key 49. KEY_F(49)
key_f50	kf50	Fe	Sent by function key 50. KEY_F(50)
key_f51	kf51	Ff	Sent by function key f51. KEY_F(51)
key_f52	kf52	Fg	Sent by function key f52. KEY_F(52)
key_f53	kf53	Fh	Sent by function key f53. KEY_F(53)
key_f54	kf54	Fi	Sent by function key f54. KEY_F(54)
key_f55	kf55	Fj	Sent by function key f55. KEY_F(55)
key_f56	kf56	Fk	Sent by function key f56. KEY_F(56)
key_f57	kf57	Fl	Sent by function key f57. KEY_F(57)
key_f58	kf58	Fm	Sent by function key f58. KEY_F(58)
key_f59	kf59	Fn	Sent by function key f59. KEY_F(59)
key_f60	kf60	Fo	Sent by function key f60. KEY_F(60)
key_f61	kf61	Fp	Sent by function key f61. KEY_F(61)
key_f62	kf62	Fq	Sent by function key f62. KEY_F(62)
key_f63	kf63	Fr	Sent by function key f63. KEY_F(63)
key_find	kfnd	@0	Find key. KEY_FIND
key_help	khlp	%1	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.

Preparing Descriptions

You can create a terminal description by copying and then modifying the description of a similar terminal. You can check the accuracy of your partial descriptions with the vi editor. Some terminals may reveal bugs in the vi editor as well as deficiencies in the ability of the **terminfo** database to provide a terminal description.

To test a new terminal description, set the **TERMINFO** environment variable to the path name of the directory containing the compiled description on which you are working. Programs then check that directory instead of the **/usr/share/lib/terminfo** directory.

To test for correct padding (if known), do the following:

1. Edit the **/etc/passwd** file at 9600 baud.
2. Delete about 16 lines from the middle of the screen.
3. Press the u key several times quickly.

   If the terminal fails to display the result properly, more padding is usually needed. You can perform a similar test for insert character.

   Note: Excessive padding slows down the terminal.

Basic Capabilities

This section describes some basic terminal capabilities. If a terminal supports one of these capabilities, the terminal's **terminfo** source file entry indicates it. The following list is a list of basic 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 Moves the cursor to the right one space.

cuu1 Moves the cursor up one space.

cud1 Move the cursor down one space.

hc Specifies a printing terminal with no softcopy unit. You should also specify the **os** capability.

ind Scrolls text up.

lf Specifies a line-feed.

lines Specifies the number of lines on a cathode ray tube (CRT) terminal.

nel Specifies a newline. The terminal behaves as if it received a carriage return followed by a line feed.

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.

ri Scrolls text down.

If the LINES and COLUMNS environment variables are set, these variables override the values in the **terminfo** database.

The local cursor motions encoded in the **terminfo** database files 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) or to go up locally off the top.

To scroll text up, a program should go to the bottom left corner of the screen and send the index string. To scroll text down, a program goes to the top left corner of the screen and sends the reverse index string. The index string is specified by the **ind** capability and the reverse index string is specified by the **ri** capability. The index string and the reverse index string are undefined when not on their respective corners of the screen.

The **am** capability determines whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply when the cursor is moved to the right (the **cuf1** capability) from the last column. A terminal has local motion from the left edge only if the **bw** capability is defined. The cursor then goes to the right edge of the previous row when moved to the left (the **cub1** capability) from the left edge. If the terminal does not have the **bw** capability, the effect is undefined, which is useful for drawing a box around the edge of the screen, for example.

A terminal has switch-selectable automatic margins if the **am** capability is specified. If the terminal has a command that moves to the first column of the next line, you can define the **nel** (new-line) capability. It does not matter whether the command clears the remainder of the current line. Therefore, if the terminal has no **cr** and **lf**, a working **nel** can still be crafted 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, xon,
```

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 are described by parameterized string capabilities. These strings have escape sequences similar to the **printf %x** format. For example, to address the cursor, you specify the **cup** capability using the row and column parameters.

The parameterized capabilities include:

cub1 Backspaces the cursor one space.

cup Addresses the cursor using the row and column parameters. 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**) or vertical absolute (**vpa**).

Sometimes the **hpa** and **vpa** capabilities are shorter than the more general two-parameter sequence and you can use them in preference to the **cup** capability. Parameterized local motions (such as, a move of *n* spaces to the right) are defined with the **cud**, **cub**, **cuf**, and **cuu** capabilities, with a single parameter indicating how many spaces to move. These capabilities are primarily useful if the terminal does not have **cup** capability.

ind*n* and **ri***n* Scrolls text. These are parameterized versions of the basic **ind** and **ri** capabilities. The *n* value is a number of lines.

mrcup Indicates the terminal has memory-relative cursor addressing.

The parameter mechanism uses a stack and has special **%** (percent sign) codes to manipulate the stack. 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).

%[[:] *Flags*] **[***Width* **[.***Precision*] **]] [doxXs]** As in the **printf** command, flags are the [- + #] and space. **%d** Prints pop() as in the **printf** command (numeric string from stack). **%2d** Prints pop() like **%2d** (minimum 2 digits output from stack). **%3d** Prints 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** Prints pop() gives **%c** (character output from stack). **%s** Prints pop() gives **%s** (string output from stack). **%p[***i***]** Pushes the *i*th parameter onto the stack where *i* is a number between 1 and 9. **%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*. **%l** Push strlen (pop( )) **%+ %- %* %/ %m** Arithmetic operators (**%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 in the following example, where **ci** denotes conditions and **bi** bodies.

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

Binary operations are in postfix form with the operands in the usual order. That is, to get `x - 5` use `%gx%{5}%-`.

If you use the - (minus sign) flag with `%[doxXs]`, then you must place a colon between the % (percent sign) and the - (minus sign) to differentiate the flag from the `%-` binary operation, for example, `%:-16.16s`.

Consider a terminal that needs to be sent `\E&a12c03Y` padded for 6 milliseconds to get to row 3 and column 12. Here the order of the rows and columns is inverted, and the row and column are zero-padded as two digits. Thus, the **cup** capability of this terminal is `cup=\E&a%p2%2.2dc%p1%2.2dY$<6>`.

Some terminals need the current row and column sent, preceded by a **^T**, with the row and column encoded in binary: `cup=^T%p1%c%p2%c`. Terminals that 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** characters, since the system may change or discard them.

Note: The library routines dealing with the **terminfo** database files set terminal modes so that tabs are not expanded by the operating system; thus, **\t** (tab) is safe to send.

A final example is a terminal that uses row and column offset by a blank character: `cup=\E=%p1%'\s'%+%c%p2'\s'%+%c`. After sending `\E=`, this operation 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

The top left corner of the screen is the home position. If the terminal has a fast way to get the cursor to the home position, specify the **home** capability. Specify, a fast way of getting to the bottom left corner with the **ll** capability. This method may involve going up (**cuu1**) from the home position, but a program should never do this itself (unless **ll** does) because the effect of moving up from the home position is not certain.

> **Note:** The home position is the same as addressing (0,0) to the top left corner of the screen, not of memory.

If the terminal has row or column absolute-cursor addressing, you should specify the single **hpa** capability (horizontal position above) and the **vpa** capability (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence and you can use them instead of the **cup** capability.

If the terminal has parameterized local motions for example, it is capable of moving the cursor *n* spaces right, you can specify the **cud**, **cub**, **cuf**, and **cuu** capabilities with a single parameter indicating how many spaces to move. These capabilities are useful if the terminal does not have the **cup** capability.

Area Clears

The following capabilities 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.

el1
: Clears from the beginning of the line to the current position, inclusive. The cursor is not moved.

Scrolling

The following insert-line and delete-line capabilities are used to indicate a terminal can:

csr Change the scrolling region. This capability takes two parameters: the top and bottom lines of the scrolling region. The top line of the screen is 0. After using this capability, the cursor position is undefined. See the **sc** and **rc** capabilities in this section.

da Retain the display above the screen. If a line is deleted or the screen is scrolled, non-blank lines can be brought in at the top. This capability is usually defined for multipage terminals.

db Retain the display below the screen. If a line is deleted or the screen is reverse scrolled, the terminal can bring the non-blank lines at the bottom. This capability is usually defined for multipage terminals.

dl1 Delete the line the cursor is on. 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 Create 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.

ind Index or scroll forward. A terminal with this capability can shift the display up one line by deleting the top line and adding a blank line at the bottom.

indn Specify the number of lines to scroll forward. This capability has meaning only if the **ind** capability is also defined.

rc Restore the cursor. This capability is useful with the **csr** and **sc** capabilities.

ri Reverse scrolling. With this capability, the terminal can shift the screen down by deleting the bottom line and adding a blank line at the top.

rin Specify the number of lines to reverse scroll. This capability has meaning only if the **ri** capability also is defined.

sc Save the cursor. If defined, you can use the **sc** capability to save the cursor before using the **csr** capability. Saving the cursor is necessary because the cursor position is undefined after you use the **csr** capability. Use the **rc** capability to restore the cursor to the position it held before you used the **csr** capability.

wind Indicates the terminal has the ability to define a window as part of memory. This is a parameterized string capability with four parameters: the starting and ending lines in memory and the starting and ending columns in memory, in that order.

A terminal that has the **csr** capability can scroll part of its screen while leaving other lines above and below the region untouched. A forward scroll applied to a region deletes the top of the region, shifts, and adds a line to the bottom of the region. When finished with the scrolling region, you should use the **csr** capability to restore the scrolling region to the full screen.

Be sure you move the cursor into the scrolling region with the **cup** capability before you attempt to scroll the region. You should not move the cursor from the region until you are done with it.

Note: If you are using a terminals **csr** capability, you may also need to use the **sc** and **rc** capability.

Terminals that have **csr** defined have a destructive scrolling region. Once a line is scrolled off the screen, the terminal cannot retrieve it. A terminal with a non-destructive scrolling region can restore scrolled lines by reversing the scrolling. Unless the **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling, do not specify the **csr** capability if the terminal has non-destructive scrolling regions.

On multipage terminals, scrolling can put a line onto another page and scrolling in the opposite direction brings the line back. Similarly, deleting a line can cause a line from another page to appear on the screen. Multipage terminals should have the **da** and **db** capabilities defined so that program that use scrolling can adjust their behavior.

A few terminals can define a window as part of memory. For these types of terminals, all clearing, deletion, insertion, and wrapping commands affect the area in memory where the window is defined.

Insert or Delete Character

Generally, terminals handle insert/delete character operations in one of two ways. 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 distinctions between typed and untyped blanks on the screen. When inserting a character, the displayed data is shifted and an untyped blank is eliminated. Once all the untyped blanks are eliminated, the displayed data wraps to the next line if you continue to insert characters. When deleting a character, an untyped blank is added to the line to compensate for the deleted character.

Generally, terminals insert/delete characters in one-line mode or multiline mode. The two types of terminals also handle untyped spaces differently. One-line mode is the most common mode. In one-line mode, insert/delete character operations affect only the characters on the current line. Insertions shift characters to the right and off the line.

Multiline mode terminals can affect more than one line. In this mode, the terminal makes a distinction between typed and untyped blanks on the screen. Inserting a character on a multiline mode terminal shifts the displayed data and eliminates untyped blanks. If all the untyped blanks are eliminated and you continue to insert characters, the display wraps to the next line. When deleting a character, multiline terminals add an untyped blank to the line to compensate for the deleted character.

Determining Your Terminal’s Type

Clearing a screen and then typing text separated by cursor motions helps you determine the type of insert/delete operations your terminal performs. Clear the screen, then proceed as follows:

1. Type `abc    def` using local cursor movements, not spaces, between the `abc` and the `def`.
2. Position the cursor before the `abc.`
3. Place the terminal in insert mode.
4. Type a line of text. If your typing causes the `abc def` characters to shift right 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.

Although these two attributes (one-line versus multiline insert mode, and different treatment of untyped spaces) are logically separate, there are no known terminals whose insert mode cannot be described with a single attribute.

Insert or Delete Character Capabilities

The **terminfo** database describes terminals that have an insert mode as well as terminals that send a simple sequence to open a blank position on the current line. The following are used to describe insert/delete character capabilities:

dch1 Deletes a single character. The **dch** capability with one parameter, *n*, deletes *n* characters.

ech Replaces the specified number of characters, starting at the cursor, with blanks. The cursor position remains unchanged.

ich1 Opens a space in a line for a character to be inserted. This sequence precedes the actual character insertion. Terminals with a true insert mode would not use 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.

rmp Indicates that padding is necessary between characters typed while not in insert mode. This capability is used in replace mode.

smdc Enters delete mode.

smir Begins insert mode.

If you are creating a **terminfo** description for a terminal that requires an insert mode and also needs a special code to precede each inserted character, then define the **smir**/**rmr**, and **ich1** capabilities. The **ich** capability, with the one parameter *n*, opens up *n* spaces so that *n* characters can be inserted.

Highlighting, Underlining, and Visual Bells

If your terminal has one or more kinds of display attributes, such as highlighting, underlining, and visual bells, you can present these in a number of ways. Highlighting, such as standout mode, presents a high-contrast, easy-to-read format that adds emphasis to error messages and other important messages. Underlining is another method to focus attention on a particular portion of the terminal. Visual bells include methods such as flashing the screen. The following capabilities describe highlighting, underlining, and visual bells:

blink Indicates terminal has blink highlighting mode.

bold Indicates terminal has extra bright highlighting mode.

civis Makes the cursor invisible.

cnorm Displays a normal cursor. This capability reverses the effects of the **civis** and **cvvis** capabilities.

cvvis Makes the cursor more visible than normal when it is not on the bottom line.

dim Indicates the terminal has half-bright highlighting modes.

eo Indicates that blanks erase overstrikes.

enacs Specifies a command string that enables alternate character set mode. Some terminals cannot enter alternate character set mode without first receiving a specific command. The **enacs** capability defines the command.

flash Indicates the terminal has a way of making the screen flash (as 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 this mode before moving the cursor or sending a new-line. Some terminals automatically leave standout mode when they move to a new line or when the cursor is addressed.

nrrmc Indicates that the **smcup** sequence does not restore the screen after a **rmcup** sequence is output. This means that you cannot restore the screen to the state prior to outputting **rmcup**.

os Indicates the terminal can overstrike an existing character without erasing the original. Overstriking creates a compound character.

prot Indicates the terminal has protected text mode. This means the terminal protects the text from overwriting or erasing. The method of protection is terminal dependent.

rev Indicates the terminal has reverse-video mode.

rmacs Exits the alternate character set mode.

rmso Exits standout mode.

rmul Ends underlining.

sgr Provides a sequence to set arbitrary combinations of attributes. The **sgr** capability can set nine attributes. In order, these attributes are the following:

standout

underline

blink

dim

bold

blank

protect

alternate character set

To turn a mode on, set it to a nonzero value. To turn a mode off, set it to 0. The **sgr** capability can only support those modes for which separate capabilities already exist on the terminal.

sgr0 Turns of all the special modes, including the alternate character set.

smacs Enters the alternate character set mode.

smcup and **rmcup** Indicate the terminal must be in a special mode when running a program that uses any of the highlighting, underlining, or visual bell capabilities. The **smcup** capability enters this mode, and the **rmcup** capability exits this mode.

This need arises, for example, with terminals having 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** capability sets the command character to be used by the **terminfo** database 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.

Highlighting, Overstriking, and Underlining

You should choose one display method as standout mode and use it to highlight error messages and other kinds of text to which you want to draw attention. For example, you could choose reverse-video plus half-bright or reverse-video alone. The sequences to enter and exit standout mode are given by the **smso** and **rmso** capabilities. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, then **xmc** should be given to tell how many spaces are left.

You should specify the **ul** boolean capability if your terminal generates underlined characters by using the underline character with no special codes. You should specify this capability even if the terminal does not otherwise overstrike characters. For terminals where a character overstriking another leaves both characters on the screen, specify the **os** capability. If the terminal can erase overstrikes with a blank, then indicate this by specifying the **eo** capability.

Example of Using the sgr Capability

The following example demonstrates how to use the **sgr** capability to turn on various modes. Assume that you must define a terminal that requires the following escape sequences to turn on various modes:

Terminfo Parameter	Mode	Escape Sequence
	none	`\E[0m`
p1	standout	`\E[0;4;7m`
p2	underline	`\E[0;3m`
p3	reverse	`\E[0;4m`
p4	blink	`\E[0;5m`
p5	dim	`\E[0;7m`
p6	bold	`\E[0;3:4m`
p7	invis	`\E[0;8m`
p8	protect	not available
p9	altcharset	`^O` (off) `^N` (on)

Note: Each escape sequence requires a 0 to turn off other modes before turning on its own mode.

You can simulate some modes by combining others. In this example, the **standout** attribute escape sequence is a combination of the **reverse** and **dim** sequences. Also, in the example the **bold** sequence is a combination of the **reverse** and **underline** sequences. To combine such modes as **underline** and **blink**, the sequence to use would be `\E[0;3;5m`.

You cannot simulate certain modes by combining others. For example, you cannot simulate the **protect** mode. In this example, the system ignores the p8 parameter. The **altcharset** mode is different in that it is either **^O** or **^N**, depending on whether the alternate character mode set is on or off. If all modes were turned on, the sequence would appear as `\E[0;3;4;5;7;8m^N`.

Some sequences are outputted for one or more modes. For example, the `;3 is` outputted when either the p2 parameter or p6 parameter is true. If you write out the above sequences along with their dependencies, the result is the following;

Sequence	**When To Output**	**terminfo Translation**
`\E[0`	always	`\E[0`
`;3`	if p2 or p6	`%?%p2%p6%|%t;3%;`
`;4`	if p1 or p3 or p6	`%?%p1%p3%|%p6%|%t;4%;`
`;5`	if p4	`%?%p4%t;5%;`
`;7`	if p1 or p5	`%?%p1%p5%|%t;7%;`
`;8`	if p7	`%?%p1%t;8%;`
`m`	always	`m`
`^N or ^O`	if p9 ^N, else ^O	`%?%p9%t^N%e^O%;`

The final result would produce a **sgr** sequence that appears as follows:

```
sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p4%t;5;%?%p1%p5%|
%t;7%;%?%p1%t;8%;m%?%p9%t^N%e^O%;,
```

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, you can define this in the **terminfo** entry for the terminal. 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.

To define the codes sent by the left-arrow, right-arrow, up-arrow, down-arrow, and home keys, use the **kcub1**, **kcuf1**, **kcud1**, and **khome** capabilities, respectively. If there are function keys such as F0, F1, ..., F63, the codes they send can be given as the **kf0**, **kf1**, ..., **kf63** capabilities. If the first eleven keys have labels other than the default F0 through F10, you can specify the labels with the **lf0**, **lf1**, ..., **lf10** capabilities. The codes transmitted by certain other special keys can be defined with:

kbs Backspace key.

kclr Clear-screen or erase key.

kctab Clear the tab stop in this column.

kdch1 Delete-character key.

kdl1 Delete-line key.

ked Clear to end of screen.

kel Clear to end of line.

khts Set a tab stop in this column.

kich1 Insert character or enter insert mode.

kil1 Insert line.

kind Scroll forward or down, or both.

kll Home down key (home is the lower left corner of the display, in this instance).

krmir Exit insert mode.

knp Next page.

kpp Previous page.

ktbc Clear-all-tabs key.

ri Scroll backward or up, or both.

In addition, if the keypad has a three-by-three array of keys including the four arrow keys, specify the other five keys as **ka1**, **ka3**, **kb2 kc1**, and **kc3**. These keys are useful when you need the effects of a three-by-three directional pad.

Strings that program function keys can be given as the **pfkey**, **pfloc**, and **pfx** capabilities. A string to program the soft screen labels can be given as **pln**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string with which to program it. Function key numbers out of this range can program undefined keys in a terminal-dependent manner. The capabilities differ in that **pfkey** causes pressing a given key to be the same as the user typing the given string, **pfloc** causes the string to be executed by the terminal in local mode, and **pfx** causes the string to be transmitted to the computer. The capabilities **nlab**, **lw**, and **lh** define the number of soft labels and the width and height. Use **smln** and **rmln** to specify the commands for turning on and off soft labels. **smln** is normally output after one or more **pln** sequences to ensure the change becomes visible.

Tabs and Initialization

If the terminal has hardware tabs, you can use **ht** capability (usually Ctrl-I) to specify the command to advance to the next tab stop. To specify the command to move left toward the previous tab stop, use the **cbt** capability. By convention, if the terminal modes indicate that operating system is expanding the tabs rather than sending them to the terminal, programs should not use the **ht** or **cbt** capabilities 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, its **terminfo** description should define the numeric capability **it** to show the number of spaces the tabs are set to. Normally, the **tput init** command uses the **it** parameter to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set.

Other, similar capabilities include the **is1**, **is2**, and **is3** initialization strings for the terminal; the **iprog** capability that specifies the terminal's initialization program, and the **if** capability that identifies the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **terminfo** file description. They are normally sent to the terminal by the **tput init** command each time the user logs in. When the user logs in, the system does the following:

- Runs the **iprog** program.
- Prints **is1**.
- Print **is2**.
- Sets the margins using the **mgc**, **smgl**, and **smgr** capabilities.
- Sets the tabs using **tbc** and **hts** capabilities.
- Prints the **if** file.
- Prints **is3**.

You can set up special terminal modes without duplicating strings by putting the common sequences in the **is2** capability and special cases in the **is1** and **is3** capabilities. To specify sequences that do a harder reset from a totally unknown state, specify the **rs1**, **rs2**, **rs3**, and **rf** capabilities that are the same as **is1**, **is2**, **is3**, and the **if** capabilities.

A few terminals use the **if** and **rf** files. However, the recommended method is to use the initialization and reset strings. These strings are output by the **tput reset** command. This command is used when the terminal starts behaving strangely or is not responding at all. Commands are normally placed in the **rs1**, **rs2**, **rs3** and **rf** capabilities only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the terminal into 80-column mode would normally be part of **is2**, but it causes an annoying screen behavior and is not necessary since most terminals initialize in 80-column mode.

If there are commands to set and clear tab stops, specify them using the **tbc** (clear all tab stops) and the **hts** (set a tab stop in the current column of every row) capabilities. If a more complex sequence is needed to set the tabs, the place the sequence in the **is2** or the **if** capability.

The **mgc** capability can clear any margin. For more information about how to set and clear margins, see Margins .

Miscellaneous Strings

If the terminal requires a character other than a null character as a pad, then specify the **pad** string. Only the first character of the **pad** string is used. If a terminal does not have a pad character, specify the **npc** capability.

If the terminal can move up or down half a line, define the **hu** (half-line up) and **hd** (half-line down) capabilities. These capabilities are primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), specify the as **ff** (usually Ctrl-L) capability.

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 following:

```
tparm(repeat_char,'x',10)
```

is the same as

```
xxxxxxxxxx
```

If the terminal has a settable command character, such as the Tektronix 4025, indicate this with the **cmdch** capability. A prototype command character is chosen that is used in all capabilities. This character is given in the **cmdch** capability to identify it. On some UNIX systems, if the **CC** environment variable exists, all occurrences of the prototype character are replaced with the character in the **CC** variable.

Terminal descriptions that do not represent a specific kind of known terminal such as switch, dialup, patch, and network, should include the **gn** (generic) capability. This capability allows programs to return errors if they cannot talk to the terminal. The **gn** capability does not apply to virtual terminal descriptions for which the escape sequences are known. If a terminal is supported by the UNIX system virtual terminal protocol, use the **vt** capability to define its terminal number.

If a terminal uses xon/xoff handshaking for the flow control, its description should include the **xon** capability. You should still include padding information as well so that routines can make better decisions about costs. However, actual pad characters are not transmitted. To specify sequences to turn on and off xon/xoff handshaking, use the **smxon** and **rmxon** capabilities. If the characters used for handshaking are not ^S and ^Q, use the **xonc** and **xoffc** capabilities to define them.

If a terminal has a meta key that acts as a shift key to set the eighth bit of any character transmitted, identify the key with the **km** capability. Otherwise, software assumes 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 **rmm** capabilities.

If a terminal has more lines of memory than fit on the screen at once, use the **lm** capability to define the number of lines of memory. A value of **lm#0** indicates that the number of lines is not fixed, but that there are still more lines of memory than fit on the screen.

Media copy strings that control an auxiliary printer connected to the terminal are identified with the following capabilities:

mc0 Prints the contents of the screen

mc4 Turns off the printer, and

mc5 Turns on the printer. When the printer is on, all text sent to the terminal is sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on.

mc5p Leaves the printer on for a specified number of characters and then turns the printer off. The parameter passed to **mc5p** should not exceed 255.

If the terminal screen does not display the text when the printer is on, specify the **mc5i** capability to signify a silent printer. All text, including the **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Status Lines

You can use the **terminfo** entry to indicate that the terminal has an extra status line that is not normally used by software,. If the status line is viewed as an extra line below the bottom line, into which the cursor can be addressed 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 **fsl** capabilities, respectively. (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** capability takes one parameter, which is the column number of the status line to which the cursor is to be moved.

If escape sequences and other special commands, such as tab, work while in the status line, specify the **eslok** capability. 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** capabilities. The status line is normally assumed to be the same width as the rest of the screen, such as **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), the width, in columns, can be indicated with the **wsl** numeric parameter.

Line Graphics

If the terminal has a line drawing alternate character set, specify the mapping of glyph to character in the **acsc** capability. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T4410v1 terminal. Use the following to define the string:

Glyph Name	**vt100+ Character**
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	-
diamond	’
check board (stipple)	a
degree symbol	f
plus or minus sign	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee	t
right tee	u
bottom tee	v
top tee	w
vertical line	x
bullet	~

The best way to describe a new terminal’s line graphics set is to add a third column to the above table with the characters for the new terminal that would produce the appropriate glyph when the terminal is in alternate character set mode. For example:

glyph name	vt100 character	tty character
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Then, you specify the **acsc** capability by specifying the characters from left to right as follows:

```
acsc=lRmFkTjGq\,x.
```

Color Manipulation

There are two methods of color manipulation, the HP method and the Tektronix method. Most existing color terminals belong to one of these two classes. The Tektronix method uses a set of *N* predefined colors (usually 8) from which a user can select *current* foreground and background colors. Thus, the terminal can support up to *N* colors mixed into *N****N* color-pairs that are displayed on the screen at the same time.

The HP method restricts the user from both defining the foreground independently of the background or the background independently of the foreground. Instead, the user must define an entire color-pair at once. Up to *M* color-pairs, made from 2**M* different colors, can be defined this way.

The numeric variables **colors** and **pairs** define the number of colors and color-pairs that the terminal can display on the screen at one time. If a terminal can change the definition of a color, you should specify the **ccc** capability. To change the definition of a color using the Tektronix method, use the **initc** capability. This capability requires four parameters: a color number ranging from 0 to colors-1 and three Red, Green, Blue (RGB) values ranging from 0 to 1,000.

Tektronix 4100 series terminals use a type of color notation called HLS (Hue Lightness Saturation) instead of RGB color notation. For such terminals, you should define the **hls** boolean capability. The last three arguments to the **initc** capability would then be HLS values where H ranges from 0 to 360 and L and S range from 0 to 100.

> **Note:** If a terminal can change the definitions of colors but uses a color notation different from RGB or HLS, you must develop a mapping to either RGB or HLS.

To set current foreground and background to a given color, use the **setf** and **setb** capabilities. These capabilities require a single parameter that specifies the number of the color. To use the HP method to initialize a color-pair, use the **initp** capability. This capability requires seven parameters:

- the number of the color-pair in the range of 0 to pairs -1
- three RGB values for the foreground
- three RGB values fro the background

When you use the **initc** or **initp** capabilities, be sure you specify the values in the order red, green, blue or hue, lightness, saturation, respectively. To make a color-pair current, use the **scp** capability. This capability takes one parameter, the number of the color-pair.

Some terminals erase areas of the screen with the current background color. In such cases, define the **bce** capability. The **op** capability contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, the **oc** capability contains a control sequence for setting all colors or -pairs to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. You should pack information about these video attributes into the **ncv** capability. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence:

Attribute	NCV Bit Number
A_STANDOUT	0
A_UNDERLINE	1
A_REVERSE	2
A_BLINK	3
A_DIM	4
A_BOLD	5
A_INVIS	6
A_PROTECT	7
A_ALTCHARSET	8

When a particular video attribute should not be used with colors, the corresponding **ncv** bit should be set to 1. Otherwise, set the bit to 0. For example, if the terminal uses colors to simulate reverse video and bold, bits 2 and 5 should be set to 1. The resulting values for **ncv** will be 22.

Special Cases

Some terminals require special support by the **terminfo** database. These terminals are not deficient. These terminals have hardware that may be slightly different than what the **terminfo** database expects of most terminals. Some of the special cases are discussed in this section. The programmer's manual for a terminal should provided all the information you need to code a **terminfo** description for the terminal.

For terminals that do not allow the ~ (tilde) character, use the **hz** capability.

Descriptions of terminals that ignore a line-feed character immediately after an **am** wrap should include the **xenl** capability. Those terminals whose cursor remains on the right-most column until another character is received rather than wrapping immediately upon receiving the right-most character, should also use the **xenl** capability.

If **el** capability is required to get rid of standout (instead of merely writing normal text on top of it), then you should specify **xhp** capability.

Terminals for which tabs change all moved characters into blanks should indicate the **xt** capability (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.

A terminal that is unable to correctly transmit the ESC (escape) or Ctrl-C characters should specify the **xsb** capability, 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.

Similar Terminals

If two terminals are very similar, you can define one as being just like the other with the **use** string capability. You can also use all of the definitions from an existing description and identify exceptions. The capabilities given before the **use** capability override those in the terminal type called by the **use** capability. To cancel a capability place `xx@` to the left of the **use** capability definition, where `xx` is the capability. For example, the entry:

```
term-nl | Terminal smkx@, rmkx@, use=term
```

defines a terminal that does not have either the **smkx** or the **rmkx** capability, and hence does not turn on the function key labels when in visual mode. This is useful for different terminal modes or for different user preferences. You can specify more than one **use** capability.

Printer Capabilities

The **terminfo** database allows you to define the capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under Terminal Capabilities that the list the capabilities by variable and by capability name.

Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that **terminfo** designers create strings that expect rounded numeric values. Programmers should always round values to the nearest integer before using them with a parameterized string capability.

Printer Resolution

A printer’s resolution is the smallest spacing of characters it can achieve. In general, printers have independent resolution horizontally and vertically. To determine the vertical resolution of a printer, measure the smallest achievable distance between consecutive printing baselines. To determine the horizontal resolution, measure the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

The **terminfo** database assumes all printers are capable of printing with a uniform horizontal and vertical resolution. The **terminfo** database currently interacts with printers as if they print inside a uniform matrix. All characters are printed at fixed positions relative to each cell in the matrix. Furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution.

Many printers are capable of proportional printing where the horizontal spacing depends on the size of the last character printed. The **terminfo** database does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of moving to a position that is an integral multiple of the smallest distance away from a previous position. Thus, printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length of width of a single page.

Some printers can have different resolutions depending on different modes. In normal mode, the existing **terminfo** capabilities are assumed to work on columns and lines, just like a video terminal. For example, the old **lines** capability specify the length of a page in lines, and the **cols** capability specifies the width of a page in columns. In **micro** mode many **terminfo** capabilities work on increments of lines and columns. With some printers, the **micro** mode may exist concurrently with **normal** mode, so that all the capabilities work at the same time.

Specifying Printer Resolution

You can specify a printer's printing resolution with several different capabilities. Each capability specifies distance in a different way. The following capabilities define print resolution:

Capability	Defined as
orhi	steps per inch horizontally
orvi	steps per inch vertically
orc	steps per column
orl	steps per line

When printing in normal mode, each character printed causes the printer to move to the next column, except in special cases described later. The distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position. The vertical distance moved is the same as the per-line resolution. When printing in micro mode, these distances can be different, and may be zero for some printers. The following specify printer resolution automatic motion after printing:

Capability	Defined as
orc	Steps moved horizontally in normal mode.
orl	Steps moved vertically in normal mode.
mcs	Steps moved horizontally in micro mode.
mls	Steps moved vertically in micro mode.

Some printers can print wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related.

If the distance moved for a regular character is the same in normal mode or micro mode (**mcs**=**ocs**), then the distance moved for a wide character is also the same in both modes. This does not mean the normal character distance is necessarily the same as the wide character distance, just that the distances do not change with a change from normal to micro mode. Use the **widcs** capability to specify the printer resolution when the automatic motion after printing a wide character is the same in both normal or micro mode.

If the distance moved for a regular character is different in micro mode from the distance moved in normal mode (**mcs<orc**), you can assume the micro mode distance is the same for a wide character printed in micro mode. In this case, you use the **mcs** capability to specify the distance moved. The printer uses the value you specify for both regular and wide characters

A printer may use control sequences to change the number of columns per inch (the character pitch) and to change the number of lines per inch (the line pitch). If these are used, the resolution of the printer changes but the type of change depends on the printer.

Capability	Defined as
cpi	Change character pitch.
cpix	If set, **cpi** changes **orhi**, otherwise the **cpi** capability changes the **orc** value.
lpi	Change line pitch
lpix	If set, **lpi** changes the **orvi** value, otherwise the **orl** value is changed.
chr	Changes steps per column.
cvr	Changes steps per line.

The **cpi** and **lpi** string capabilities have a single argument, the pitch in columns (or characters) and lines per inch, respectively. The **chr** capability and **cvr** string capabilities each have a single argument, the number of steps per column and line, respectively.

Using any of the control sequences in these strings implies a change in some of the values of the **orc**, **orhi**, **orl**, and **orvi** capabilities. Also, the distance moved when a wide character is printed, specified by the **widcs** capability, changes in relation to the **orc** value. The distance moved when a character is printed in micro mode, **mcs**, changes similarly, with one exception: if the distance is 0 or 1, then no change is assumed.

Programs that use the **cpi, lpi, chr, or cvr** capability should recalculate the printer resolution and should recalculate other values. For more information, see Effect of Changing Printing Resolution .

The figure Specification of Printer Resolution Effects of Changing the Character/Line Pitches shows the effects on printer resolution before and after a change.

*V*cpi, *V*lpi, *V*chr, and *V*cvr are the arguments used with **cpi**, **lpi**, **chr**, and **cvr** respectively. The dagger symbol indicates the old value.

Capabilities that Cause Movement

In the following descriptions, *movement* refers to the motion of the *current position*. With video terminals this would be the cursor; with some printers this is the carriage position. Other printers have different equivalents. In general, the current position is where a character would be displayed if printed.

The **terminfo** database has string capabilities for control sequences that cause movement a number of full columns or lines. It also has equivalent string capabilities for control sequences that cause movement a number of small steps. The following are the string capabilities for motion:

Capability	Description
mcub1	Move 1 step left.
mcuf1	Move 1 step right.
mcuu1	Move 1 step up.
mcud1	Move 1 step down.
mcub	Move *N* steps left.
mcuf	Move *N* steps right.
mcuu	Move *N* steps up.
mcud	Move *N* steps down.
mhpa	Move *N* steps from the left.
mvpa	Move *N* steps from the top.

The last six strings are each used with a single *N* argument.

Sometimes the motion is limited to less than the width or length of a page. Also, some printers do not accept absolute motion to the left of the current position. The following capabilities limit motion:

Capability	Description
mjump	Limits the use of **mcub1**, **mcuf1**, **mcuu1**, and **mcud1** capabilities.
maddr	Limits the use of the **mhpa** and **mvpa** capabilities.
xhpa	If set, the **hpa** and **mhpa** capabilities are negated.
xvpa	If set, the **vpa** and **mvpa** capabilities are negated.

If a printer needs to be in *micro mode* for the motion capabilities to work, you can define a string capability to contain the control sequence to enter and exit micro mode. A boolean is available for those printers where using a carriage return causes an automatic return to normal mode. The following capabilities are related to micro mode behavior:

Capability	Description
smicm	Enter micro mode.
rmicm	Exit micro mode.
crxm	Using the key specified by the **cr** capability exits micro mode.

The movement made when a character is printed in the rightmost position varies among printers. Some make no movement, some move to the beginning of the next line, others move to the beginning of the same line. The **terminfo** database has boolean capabilities that description all three cases. The **sam** capability specifies that the printer automatically moves to the beginning of the same line after the character is printed in the rightmost margin.

Some printers can be put in a mode where the normal direction of motion is reversed. This mode is especially useful when there exists no capabilities for leftward or upward motion, you can build these capabilities from the motion reversal capability and the rightward or downward motion capabilities. It is best to leave it up to an application to build the leftward or upward capabilities, though, and not enter them into to the **terminfo** database. This allows several reverse motions to be strung together without intervening wasted steps that leave and reenter reverse mode. The following capabilities control entering and exiting reverse modes:

Capability	Description
slm	Reverse sense of horizontal motions.
rlm	Restore sense of horizontal motions.
sum	Reverse sense of vertical motions.
rum	Restore sense of vertical motions.

The following capabilities affect the screen while the horizontal motions are reversed:

Capability	Description
mcub1	Move 1 step right.
mcuf1	Move 1 step left.
mcub	Move *N* steps right.
mcuf	Move *N* steps left.
cub1	Move 1 column right.
cuf1	Move 1 column left.
cub	Move *N* columns right.
cuf	Move *N* columns left.

The following capabilities affect the screen whilethe vertical motions are reversed:

Capability	Description
mcuu1	Move 1 step down.
mcud1	Move 1 step up.
mcuu	Move *N* steps down.
mcud	Move *N* steps up.
cuu1	Move 1 line down.
cud1	Move 1 line up
cuu	Move *N* lines down.
cud	Move *N* lines up.

The reverse motion mode should not affect the **mvpa** and **mhpa** absolute motion capabilities. The reverse vertical motion mode should, however, also reverse the action of the line *wrapping* that occurs when a character is printed in the right-most position. Thus printers that have the standard **terminfo** capability **am** defined should move to the beginning of the previous line when a character is printed on the right-most position and the printer is in reverse-vertical motion mode.

The action when any other motion capabilities are used in reverse motion modes is not defined. Thus, programs must exit reverse motion modes before using other motion capabilities.

Two miscellaneous capabilities complete the list of new motion capabilities, the **docr** and the **zerom** capability. The **docr** capability provides a list of control characters that cause a carriage return. This capability is useful for printers that move the current position to the beginning of a line when certain control characters, like line-feed or form-feed are used. The **zerom** capability prevents automatic motion after printing a single character. This capability suspends the motion that normally occurs after printing a character.

Margins

The **terminfo** database provides two strings for setting margins on terminals: one for the left and one for the right margin. Printers, however, have two additional margins for the top and bottom margins of each page. Furthermore, some printers do not require using motion strings to move the current position to a margin and fixing the margin there, as with existing capabilities, but require the specification of where a margin should be regardless of the current position. Therefore, the **terminfo** database offers six additional strings for defining margins with printers. The following capabilities affect margins:

Capability	Definition
smgl	Set left margin at the current column.
smgr	Set right margin at the current column.
smgb	Set the soft bottom margin at the current line.
smgt	Set the soft top margin at the current line.
smgbp	Set the soft bottom margin at line *N*.
smglp	Set the soft left margin at column *N*.
smgrp	Set the soft right margin at column *N*.
smgtp	Set soft top margin at line *N*.

The last four strings are used with a single *N* parameter. This parameter specifies a line or column number, where 0 is the top line and column 0 is the left-most column.

Note: Not all printers use 0 for the top line or the left-most column.

All margins can be cleared with the **mgc** capability.

Shadows, Italics, Wide Characters, Superscripts, and Subscripts

Five new sets of strings are used to describe the capabilities that printers have of enhancing printed text. The following define enhanced printing capabilities:

Capability	Definition
sshm	Enter shadow-printing mode.
rshm	Exit shadow-printing mode.
sitm	Enter italicizing mode.
ritm	Exit italicizing mode.
swidm	Enter wide-character mode.
rwidm	Exit wide-character mode.
ssupm	Enter superscript mode.
rsupm	Exit superscript mode.
supcs	List of characters available as superscripts.
ssubm	Enter subscript mode.
rsubm	Exit subscript mode.
subcs	List of characters available as subscripts.

If a printer requires the **sshm** control sequence before every character to be shadow-printed, the **rshm** string is left blank. Thus, programs that find a control sequence in **sshm** but none in shadow printing mode should use the control sequence specified by the **sshm** capability before every character to be shadow printed. Otherwise, the control sequence should be used once before the set of characters to be shadow-printed, followed by exiting shadow-printing mode.

The **terminfo** database also has a capability for printing emboldened text, the **bold** capability. While shadow printing and emboldened printing are similar in that they darken the text, many printers produce these two types of print in slightly different ways. Generally emboldened printing is done by overstriking the same character one or more times. Shadow printing likewise usually involves overstriking, but with a slight movement up and/or to the side so that the character is fatter.

It is assumed that enhanced printing modes are independent modes, so that it would be possible, for instance, to shadow print italicized subscripts.

As mentioned earlier, the amount of motion automatically made after printing a wide character should be given in the **widcs** capability.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in the **supcs** or **subcs** capabilities, respectively. If the **ssupm** or **ssubm** strings contain control sequences, but the corresponding **supcs** or **subcs** strings are empty, it is assumed that all printable ASCII characters are available as superscripts or subscripts.

Automatic motion made after printing a superscript or subscript is assumed to be the same as for regular characters. For example, printing any of the following result in equivalent motion:

```
Bi Bi Bi
```

The boolean capability **msgr** describes whether an application can use motion control sequences while in standout mode. This capability is extended to cover the enhanced printing modes added here. The **mgsr** capability should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if the **mgsr** capability is not set, a program should end these modes before attempting any motion.

Alternate Character Sets

In addition to allowing you to define line graphics, the **terminfo** database also lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets:

Capability	Definition
scs	Select character set *N*. The *N* parameter specifies a number from 0 to 63 that identifies a character set.
scsd	Start definition of character set *N*, *M* characters. The *N* parameter specifies a number from 0 to 63 that identifies a character set and the *M* parameter specifies the number of characters in the set.
defc	Defines a character *A* to be *B* dots wide with a descender *D*. The *A* parameter is the ASCII code representation for the character. The *B* parameter specifies the width of the character in dots. The *D* parameter specifies whether the character is a descender or not. If the character is a descender, specify a 1 for the *D* parameter. Otherwise, specify a 1. This string is followed by a string of image-data bytes that describe how the character looks.
rcsd	End definition of character set *N*. The *N* parameter specifies a number from 0 to 63 that identifies a character set.
csnm	List of character set names.
daisy	Indicates the printer has manually changed print-wheels.

Character set 0 is the default character set. This is the set that is present after the printer is initialized. Not every printer supports 64 character sets. If you specify a set that a printer does not support, the **tparm** subroutine returns a null result.

If your application must define a character before using it, use the **scsd** control sequence before defining the character set, and the **rcsd** after. If you specify an invalid character set for either of these capabilities, the **tparm** subroutine returns a null resolution. If your application must select a character set after it is defined, the **scs** control sequence should follow the **rcsd** control sequence. By examining the results of using each of the **scs**, **scsd**, and **rcsd** strings with a character set number in a call to the **tparm** subroutine, a program can determine which of the three are needed.

Between use of the **scsd** and **rcsd** strings, the **defc** string should be used to define each character. To print any character on printers defined in the **terminfo** database, the ASCII cod is sent to the printer. This is true for characters in an alternate set as well as *normal* characters. Thus, the definition of a character includes the ASCII code that represents it. In addition, the width of the character includes the ASCII code that represents it. In addition, the width of the character in dots is given, along with tan indication of whether the character is a descender. A descender is a character whose shape extends below the baseline, for example the character g is a descender. The width of the character is dots also indicates the number of image-data bytes that will follow the **defc** string. These image-data bytes indicate where in a dot-matrix pattern ink should be applied to *draw* the character. The number of these bytes and their form are defined below under Dot-Mapped Graphics.

It is easiest for the creator of **terminfo** entries to refer to each character set by number. However, these numbers will be meaningless to the application developer. The **csnm** capability alleviates this problem by providing names for each number.

When used with a character set number in a call to the **tparm** subroutine, the **csnm** capability produces the equivalent name. Use these names as a references only. No naming convention is implied, although anyone who creates a **terminfo** entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the **csnm** string to determine the correct number), or by name, where the application examines the **csnm** capability to determine the corresponding character set number.

The alternate character set capabilities are likely to be used only with dot-matrix printers. If they are not available, do not define these strings. For printers that have manually changed print-wheels or font cartridges, set the boolean **daisy** capability.

Dot-Matrix Graphics

Dot-matrix printers typically have the capability to reproduce raster-graphics images. Three new numeric capabilities and three new string capabilities can help a program draw raster-graphic images independent of the type of dot-matrix printer or the number of pins or dots the printer can handle at one time. The dot-matrix capabilities are as follows:

Capability	Definition
npins	Number of pins *N* in the print-head. The *N* parameter specifies the number of pins.
spinv	Spacing of pins vertically in pins per inch.
spinh	Spacing of dots horizontally in dots per inch.
porder	Matches software bits to print-head pins.
sbim	Start printing bit image graphics, *B* bits wide. The *B* value specifies the width of the image in dots.
rbim	End printing bit image graphics.

The model of dot-matrix or raster-graphics that the **terminfo** database presents is similar to the technique used for most dot-matrix printers. Each pass of the printer's print-head is assumed to produce a dot-matrix that is *N* dots high and *B* dots wide. This is typically a wide, squat, rectangle of dots. The height of this rectangle in dots varies from one printer to the next. This is given in the **npins** numeric capability. The size of the rectangle in fractions of an inch will also vary. The size can be deduced from the **spinv** and **spinh** numeric capabilities. With these three values an application can divide a complete raster-graphics image into several horizontal strips, perhaps interpolating to account for different dot spacing vertically and horizontally.

The **sbim** and **rbim** capabilities are used to start and end a dot-matrix image, respectively. The **sbim** capability is used with a single argument that gives the width of the dot-matrix in dots. A sequence of image-data bytes are sent to the printer after the **sbim** capability and before the **rbim** string. The number of bytes is an integral multiple of the width of the dot-matrix. The multiple and the form of each byte is determined by the **porder** capability is described below.

The **porder** capability is a comma-separated list of pin numbers. The position of each pin number in the list corresponds to a bit in a data byte. The pins are numbered consecutively from 1 to **npins**, with 1 being the top pin. The term pin is used loosely here. Ink-jet dot matrix printers don't have pins but they do have an equivalent method of applying a single dot of ink to paper. The bit positions in **porder** are in groups of 8, with the first position in each group the most significant bit and the last position the least significant bit.

The image-data bytes are computed from the dot-matrix image, mapping vertical dot positions in each print-head pass into eight-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. If a position is skipped in **porder**, a 0 bit is used. There must be a multiple of 8 bit positions used or skipped in **porder**. If not, 0 bits are used to fill the last byte in the least significant bits.

Effect of Changing Printing Resolution

If the control sequences to change the character pitch or the line pitch are used, the pin or dot spacing may change. The following capabilities change pitch on dot-matrix graphics:

Capabilities	Definition
cpi	Change the character pitch.
cpix	If set, **cpi** changes **spinh**.
lpi	Change line pitch.
lpix	If set, **lpi** changes **spinv**.

Programs that use **cpi** or **lpi** should recalculate the dot spacing. The figure Dot-Matrix Graphics Effects of Changing the Character/Line Pitches shows graphics both before and after a change in pitch.

The **orhi’** and **orhi** values are the values of the horizontal resolution in steps per inch, before using **cpi** and after using **cpi**, respectively. Likewise, **orvi’** and **orvi** are the values of the vertical resolution in steps per inch, before using **lpi** and after using **lpi**, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.

Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near letter-quality printing or draft-quality printing. Usually, it is important to be able to choose one or the other because the rate of printing generally falls off as the quality improves. The capabilities that specify print quality are the following:

Capability	Definition
snlq	Set near-letter quality print.
snrmq	Set normal quality print.
sdrfq	Set draft-quality print.

The capabilities are listed in decreasing levels of quality. If a printer does not have all three levels, one or two of the strings should be left blank as appropriate.

Printing Rate and Buffer Size

Because there is no standard protocol that synchronizes a printer with a program, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has printed. Two new numeric capabilities can help a program estimate what has printed, the **cps** and **bufsz** capabilities.

The **cps** capability specifies the nominal print rate in characters per second. The **cps** capability is the nominal or average rate at which the printer prints characters. If this value is not given, estimate the rate at one-tenth the prevailing baud rate.

The **bufsz** capability defines a terminal's buffer capacity in characters. The **bufsz** value is the maximum number of subsequent characters buffered before the guaranteed printing of an earlier character, assuming proper flow control was used. If this value is not given it is assumed that the printer does not buffer characters, but prints them as they are received.

As an example, if a printer has a 1000-character buffer, then sending the letter "a" followed by 1000 additional characters is guaranteed to cause the letter "a" to print. If the same printer prints at the rate of 100 characters per second, then it should take 10 seconds to print all the characters in the buffer, less if the buffer is not full. By keeping track of the characters sent to a printer, and knowing the print rate and buffer size, a program can synchronize itself with the printer.

Most printer manufacturers advertise the maximum print rate, not the nominal print rate. A good way to get a value to put in for **cps** is to generate a few pages of text, count the number of printable characters, then see how long it takes to print the text.

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in **cps**. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in **cps**. If the application is using **cps** to decide how long it should take a printer to print a block of text, the application should pad the estimate. If the application is using **cps** to decide how much text has already been printed, it should shrink the estimate. The application errs in favor of the user, who wants, above all, to see all the output in its correct place.

Database File Names

Compiled **terminfo** file descriptions are placed in subdirectories under the **/usr/share/lib/terminfo** directory 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/share/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 description for the terminal `term4-nl` can be found in the file **/usr/share/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 **terminfo** entry describes a terminal:

```
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,
```

The following **terminfo** entry describes a terminal:

```
ibm3161|ibm3163|wy60-316X|wyse60-316X|IBM 3161/3163 display,
        am,             mir,            cr=^M,          ind=^J,
        cols#80,        it#8,           lines#24,
kich1=\EP\040\010,
        ed=\EJ,         el=\EI,         cup=\EY%p1%’ ’%+%c%p2%’
’%+%c,
        clear=\EH\EJ,   dch1=\EQ,       dl1=\EO,        cud1=\EB,
        cub1=\ED,       blink=\E4D,     bold=\E4H,
sgr0=\E4@\E<@,
        invis=\E4P,     rev=\E4A,       cuf1=\EC,
rmso=\E4@,
        smso=\E4A,      rmul=\E4@,      cuu1=\EA,
smul=\E4B,
        sgr=\E4%’@’%?%p1%t%’A’%|%;
                   %?%p2%t%’B’%|%;
                   %?%p3%t%’A’%|%;
                   %?%p4%t%’D’%|%;
                   %?%p5%t%’@’%|%;
                   %?%p6%t%’H’%|%;
                   %?%p7%t%’P’%|%;%c
                   %?%p9%t\E>A%e\E<@%;,
        box1=\354\361\353\370\352\355\367\365\366\364\356,
        box2=\354\361\353\370\352\355\367\365\366\364\356,
batt2=md,
```

```
        ktbc=\E\0401,   kil1=\EN,       kbs=^H,
kclr=\EL^M,
        kcud1=\EB,      kdch1=\EQ,      kel=\EI,
khome=\EH,
        kcub1=\ED,      kdl1=\EO,       ktab=^I,        kcbt=\E2,
        kcuf1=\EC,      ked=\EJ,        kctab=\E1,      khts=\E0,
        kcuu1=\EA,      knl=\r,         kact=\E8\r,
        kf1=\Ea\r,      kf2=\Eb\r,      kf3=\Ec\r,
kf4=\Ed\r,
        kf5=\Ee\r,      kf6=\Ef\r,      kf7=\Eg\r,
kf8=\Eh\r,
        kf9=\Ei\r,      kf10=\Ej\r,     kf11=\Ek\r,
kf12=\El\r,
        kf13=\E!a\r,    kf14=\E!b\r,    kf15=\E!c\r,
kf16=\E!d\r,
        kf17=\E!e\r,    kf18=\E!f\r,    kf19=\E!g\r,
kf20=\E!h\r,
        kf21=\E!i\r,    kf22=\E!j\r,    kf23=\E!k\r,
kf24=\E!l\r,
        smcup=\E>A,     rmcup=\E>A,     msgr,
        home=\EH,       bel=^G, mc5=^P^R, mc4=^P^T,
```

Implementation Specifics

The **terminfo** database is part of Base Operating System (BOS) Runtime.

Files

/usr/share/lib/terminfo/?/*	Compiled terminal capability database.

Related Information

The **captoinfo** command, **infocmp** command, **tic** command.

The **printf**, **fprintf**, or **sprintf** subroutine.

Curses Overview for Programming in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*.

.tiprc File Format for tip

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 searches 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.

Examples

Following is a sample **.tiprc** file:

```
be
ba=9600
!echocheck
```

This file toggles the **beautify** (`be`) variable, sets the **baudrate** (`ba`) variable to `9600`, and resets the **echocheck** variable to the default setting.

Implementation Specifics

This file is part of the Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

$HOME/.tiprc

Specifies the complete path name of the **.tiprc** file.

Related Information

The **tip** command.

trcfmt File Format

Purpose

Stores trace templates.

Description

The **trcrpt** command, which formats trace reports, uses trace templates to determine how the data contained in trace entries should be formatted. All trace templates are stored in the master template file, **/etc/trcfmt**. Trace templates identify the trace hook ID, the version and release number, the indentation level, the event label, and data description fields. The data description fields contain formatting information for the trace entry data and can be repeated as many times as is necessary to format all of the trace data in the trace entry.

Modifying this File

The **trcfmt** file should only be modified using the **trcupdate** command. Trace hooks with values less than 010 are for internal use by the trace facilities. If these hooks are changed, the performance of trace, in particular **trcrpt**, is unpredictable.

Trace Entries

The data recorded for each traced event consist of a word containing the trace hook identifier and the hook type followed by a variable number of words of trace data optionally followed by a timestamp. The word containing the trace hook identifier and the hook type is call the hook word. The remaining two bytes of the hook word are called hook data and are available for recording event data.

HookWord The first two bytes of a *HookWord* contain the *HookID* and *HookType*. The contents of the second two bytes depends on the value of the *HookType*.

HookID The *HookID* is represented in the trace entry as 3 hexadecimal digits. For user programs, the hook id may be a value ranging from `0x010` to `0x0FF`. *HookID*s are defined in the **/usr/include/sys/trchkid.h** file.

HookType The *HookType* is a 4-bit value that identifies the format of the remainder of the trace entry. You specify the *HookType* when you record the trace entry.

Value	Trace Entry Format
1	The trace entry consists of only the *HookWord*. The third and fourth bytes of the *HookWord* contain trace data. Trace entries of this type are recorded using the **trchook** or **utrchook** subroutine.
2	The trace entry consists of the *HookWord* and one additional word of trace data. The third and fourth bytes of the *HookWord* contain trace data. Trace entries of this type are recorded using the **trchook** or **utrchook** subroutine.
6	The trace entry consists of the *HookWord* and up to five additional words of trace data. The third and fourth bytes of the *HookWord* contain trace data. Trace entries of this type are recorded using the **trchook** or **utrchook** subroutine.
8	The trace entry consists of the *HookWord* and a data word followed by a variable number of bytes of trace data and a timestamp. The third and fourth bytes of the *HookWord* contain the number of bytes of trace data which follows the trace word. Trace entries of this type are recorded using the **trcgent** subroutine or the **trcgenkt** kernel service.
9	The trace entry consists of the *HookWord* and a timestamp. The third and fourth bytes of the *HookWord* contain trace data. Trace entries of this type are recorded using the **trchook** or **utrchook** subroutine.
A	The trace entry consists of the *HookWord*, one additional word of trace data, and a timestamp. The third and fourth bytes of the *HookWord* contain trace data. Trace entries of this type are recorded using the **trchook** or **utrchook** subroutine.
E	The trace entry consists of the *HookWord*, up to five additional words of trace data, and a timestamp. The third and fourth bytes of the *HookWord* contain trace data. Trace entries of this type are recorded using the **trchook** or **utrchook** subroutine.
0	The trace entry consists of the *HookWord* and a data word followed by a variable number of bytes of trace data. The third and fourth bytes of the *HookWord* contain the number of bytes of trace data which follows the trace word. Trace entries of this type are recorded using the **trcgen** subroutine or the **trcgenk** kernel service.

Data Pointer

The DATA POINTER is a pointer to the current position in the trace entry. The DATA POINTER is changed by the **trcrpt** as it interprets the template and formats the trace entry. The initial position of the DATA POINTER is the third byte of the *HookWord* for *HookTypes* `1, 9, 2, A, 6,` and `E` and the first byte after the *HookWord* for *HookTypes* `0` and `8`.

Trace Data Formatting

Indentation Level

The formatted trace data is aligned in columns corresponding to the source of the trace event. This is identified in each template using the **L=***X* descriptor. The possible values of the **L=***X* command are as follows:

L=APPL Outputs the trace data in the APPL (application) column.

L=SVC Outputs the trace data in the SVC (system call) column.

L=KERN Outputs the trace data in the KERN (kernel) column.

L=INT Outputs the trace data in the INT (interrupt)column.

Continuation Character

A \ (backslash) at the end of a line must be used to continue a template on the next line.

Labels or Text Strings

Individual strings (or labels) can be separated by any number of spaces or tabs, but all excess spacing is compressed to one blank on the trace report unless other format structures are put into effect. Labels are enclosed in double quotes (" ").

\n Outputs to a new line. Data on the new line is left-justified according to the value set in the INDENTATION LEVEL.

\t Inserts a tab. Tabs are expanded to spaces, using a fixed tabstop separation of 8.

Format Codes

DATA POINTER Position Format Codes

G*m.n* Sets DATA POINTER to byte.bit location *m.n*.

O*m.n* Advances DATA POINTER by *m.n* byte.bits.

R*m* Decrements DATA POINTER by *m* bytes.

Output Format Codes

B*m.n*	Sends output in Binary format where *m* is the length of the data in bytes and *n* is the length in bits. Unlike the other printing format codes, the DATA POINTER can be bit aligned and is not rounded up to the next byte boundary.
D2, D4 , D8	Converts data to signed decimal format. The length of the data is two, four, or eight bytes, and the DATA POINTER is advanced by the same number of bytes.
F4	Converts data to C type ’float’ floating point format. The length of the data is 4 bytes, and the DATA POINTER is advanced by 4 bytes.
F8	Converts data to C type ’double’ floating point format. The length of the data is 8 bytes, and the DATA POINTER is advanced by 8 bytes.
S1, S2, S4	Left-justifies ASCII strings. The length of the string is in the first byte (half-word, word) of the data. The length of the string does not include this byte.
T4	Outputs the next 4 bytes as a date and time string.
U2, U4 , U8	Converts data to unsigned decimal format. The length of the data is two, four, or eight bytes, and the DATA POINTER is advanced by the same number of bytes.
X*m*	Converts data to hexadecimal format. The DATA POINTER is advanced by *m* bytes.

Interpreter Format Codes

E1, E2, E4	Outputs the next byte (half_word, word) as an ’errno’ value, replacing the numeric code with the corresponding #define name in the **/usr/include/sys/errno.h** file. The DATA POINTER is advanced by 1, 2, or 4 bytes.
P4	Uses the next word as a process ID, and outputs the pathname of the executable with that process ID. Process IDs and their pathnames are acquired by the **trace** command at the start of a trace and by the **trcrpt** command via a special EXEC tracehook. The DATA POINTER is advanced by 4 bytes.

Switch Statements

A SWITCH statement is a format code followed by a comma. Each CASE entry of the SWITCH statement consists of:

1. A *’MatchValue’* with a type (usually numeric) corresponding to the format code.
2. A simple ’*String*’ or a new ’*Descriptor*’ bounded by braces. A descriptor is a sequence of format codes, strings, switches, and loops.
3. A comma delimiter.

The switch is terminated by a CASE entry without a comma delimiter. The CASE entry is selected as the first entry whose *MatchValue* is equal to the expansion of the format code. The special matchvalue ’*’ is a wildcard and matches anything.

The DATA POINTER is advanced by the format code.

LOOP Statements

Loops are used to output binary buffers of data; therefore, the descriptor for a LOOP is usually X0 or X1. The syntax of a loop is LOOP format_code {descriptor}. The descriptor is executed *N* times, where *N* is the numeric value of the format code.

The DATA POINTER is advanced by the format code and by the operations of the descriptor.

Macros

Macros are temporary variables that work like shell variables. They are assigned a value with the syntax:

```
{{ $xxx = EXPR }}
```

where EXPR is a combination of format codes, macros, and constants. The operators + (addition), - (subtraction), / (division), and * (multiplication). are permissible within macros.

Predefined Macros

Macro Name	Description
$BASEPOINTER	Marks the starting offset into an event. The default is `0`, but the actual offset is the sum of the values of DATA POINTER and BASE_POINTER. It is used with template subroutines when the parts of an event have same structure and can be printed by same template but may have different starting points into an event.
$BREAK	Ends the current trace event.
$D1 - $D5	Dataword 1 through dataword 5. The DATA POINTER is not moved.
$DATAPOINTER	Activates the DATA POINTER. It can be set and manipulated like other user macros.
$DEFAULT	Uses the DEFAULT template 008.
$ERROR	Outputs an error message to the report and exit from the template after the current descriptor is processed. The error message supplies the logfile, the logfile offset of the start of that event, and the trace ID.
$EXECPATH	Outputs the pathname of the executable for the current process.
$HB	Number of bytes in **trcgen** subroutine variable length buffer. This is also equal to the 16-bit hook data.
$HD	Hook data (lower 16 bits).

$HT	Allows for multiple, different **trchook** subroutine call with the same template. The return values of the **$HT** macro are:

Value	Description
1	hook word
2	hook word and one additional word
6	hook word and up to five data words
9	hook word and a timestamp
A	hook word, one data word, and a timestamp
E	hook word, up to five data words, and a timestamp.

The DATA POINTER is not changed.

$L1-$L2	Long (64-bit) dataword 1, or 2. For example, **$L1** is the concatination of **$d1** and **$d2**. The 64-bit values would most likely have been traced with the **TRCHK64L1** or **TRCHK64L2** macros. No change to data pointer.
$LOGID0	Current logfile offset at the start of the event.
$LOGIDX	Current logfile offset into this event.
$LOGFILE	Returns the name of the logfile being processed.
$RELLINENO	Line number for this event. The first line starts at 1.
$PID	Outputs the current process ID.
$SKIP	Ends the current trace event without printing.
$STOP	Immediately ends a trace report.
$SVC	Outputs the name of the current system call.
$TID	Outputs the current kernel thread ID.
$TRACEID	Returns the trace ID of the current event.

Built-in Macros

The built-in macros are:

buftofilename (bp)	Looks up filename by buf struct.
fdinstall ( )	Installs the file descriptor and the current v-node from lookuppn as a file_descriptor/v-node pair for this process ID.
fdtofilename ( )	Looks up the filename for the given file descriptor for this process ID. If the filename is not found, nothing is output.
flih ( )	Advances the Interrupt Depth.
lookuppninstall1	Installs the filename as the current file with the **trcrpt** command.
lookuppninstall2	Install the v-node as the current v-node. It also installs the current_v-node/current_file as a v-node/filename par.
pfsrdwrinstall1 (vp)	Sets the current v-node of this process to vp.
pfsrdwrinstall2 (VA.S, count)	Creates a virtual address/v-node structure to be filled in be VMM hooks if a page fault occurs.
resume ( )	Decrements the Interrupt Depth.
setdelim ( )	Inhibits spaces between characters.
slihlookup ( )	Looks up the second level interrupt handler.
sidtofilename (sid)	Looks up filename by segment ID.
vmbufinstall ( )	Looks up the v-node of the file through the virtual page/sid and install the v-node and buf as a v-node/bp pair. This will be used by lvm on down.
v-nodetofilename (vp)	Looks up filenames by v-node.
vpagetofilename (vpage, sid)	Looks up filenames by vpage and segment ID.

Implementation Specifics

The **trcfmt** file is part of the operating system.

Files

/etc/trcfmt	Stores trace templates.
/usr/include/sys/trchkid.h	Defines hook identifiers.
/usr/include/sys/trcmacros.h	Defines trace macros.

Related Information

The **trcupdate** command.

The **trcgen** subroutine, **trchook** subroutine.

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* variable and are ASCII strings. Strings inside **[ ]** (brackets) are optional. The **troff** command can produce them, but they are not required for the specification of the language. The **\n** command character has the standard meaning of new-line character. Between commands, white space has no meaning. White-space 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*	Indicates the font is to be mounted in the position specified by the *Number* variable value, which ranges from 0 (zero) to the highest font currently mounted. The 0 (zero) value is a special position, called by the **troff** command, but not directly accessible by the user. Fonts are normally mounted starting at position 1 (one).
c*Character*	Generates the specified character at the current location on the page; the value specified by the *Character* variable is a single-byte character.
C*XYZ*	Generates the *XYZ* special character whose name is delimited by white space. The name is one of the special characters legal for the typesetting device as specified in the **DESC** file. This file resides in a directory specific to the typesetting device. For instruction, see troff Font File Format and the **/usr/lib/font/dev***Device* 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 single-byte 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 single-byte character, specified by the *Character* variable, 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 (zero). 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 (zero).
#...\n	Initiates a comment line with the **#** (pound sign).
Dl *X Y*	Draws a line from the current position to that specified by the *X,Y* variables.
Dc *D***\n**	Draws a circle of the diameter specified by the *D* variable 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* variable is the axis in the *X* direction and the *DY* variable 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 *DH1I+DH2, DV1+DV2* variable 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 P[aper], the I changed the followinf*PaperSize W L***/n** Specifies the name of the paper size to be printed. Valid paper sizes are Letter, Legal, A4, B5, Executive, and A5, where *W* and *L* are the paper width and length in machine units. **x i[nit]\n** Initializes the typesetting device. The actions required are dependent on the device. An initializing command always occurs before any output generation is attempted. **x T** *Device***\n** Specifies the name of the typesetter with the *Device* variable. This is the same as the variable to the **-T** flag. Information about the typesetter is found in the **/usr/lib/font/dev***Device* directory. **x r**[**es**] *N H V***\n** Specifies the resolution of the typesetting device in increments per inch with the *N* variable. The *H* variable specifies units of basic increments that horizontal motion will take place. The *V* variable indicates the units of basic increments for vertical motion. **x p**[**ause**]**\n** Pauses the process by causing the current page to finish but does not relinquish the typesetter. **x s**[**top**]**\n** Stops the process by causing 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. **x c**[**codeset**] **CS\n** Switch to codeset **CS**. For example:

```
x codeset ISO8859-1
```

The following commands are effective on multi-byte characters.

Q*C1C2*	Outputs the character specified by the 2 bytes specified by the *C1* and *C2* variables. The high-order bits can be set in these bytes.
R*C1C2C3*	Outputs the character specified by the three bytes of the *C1*, *C2*, and *C3* parameters. The high-order bits can be set in these bytes.
S*C1C2C3C4*	Outputs the character specified by the four bytes of the *C1*, *C2*, *C3*, and *C4* parameters. The high-order bits can be set in these bytes.

Implementation Specifics

This file is part of Formatting Tools in the Text Formatting System.

Files

/usr/lib/font/dev*Device* directory	Contains the **DESC** file and phototypesetter-specific files.

Related Information

International Character Support in Text Formatting Overview in *AIX Version 4.3 System User's Guide: Operating System and Devices* discusses the European-language extended character set and the commands that use it.

The **troff Font** File Format.

troff Font File Format

Purpose

Specifies 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 the *Name* variable specifies the name of the phototypesetter.

The ASCII **DESC** file in the **/usr/lib/font/dev***Name* directory within the **troff** command source directory describes the characteristics of the phototypesetter specified by the *Name* variable. 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 # (pound sign) are ignored.

For many typesetters, downloaded fonts are supported in a general fashion. The bitmaps for these fonts are stored in the **/usr/lib/font/dev***Name***/bitmaps** directory. Each font size pair is stored in a file with a name of the form *Fontname-Size*.**pk**. For example:

```
B-24.pk
```

These bitmaps are stored in the PK packed-font format used by TeX and its post-processors. These bitmaps are easily generated form readily available programs, such as METAFONT, or easily converted from other forms.

In addition to the bitmap files, a **troff** font file, as described here, is required for each font typeface. In the unitwidth field of this file, the width of each character bitmap in device units is given.

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. The value of the *Number* variable 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* — Maximum number of characters in a font.

sizes *Number1 Number2...* — List of point sizes available on typesetter, ended by 0.

fonts *NumberName...* — Number of initial fonts, followed by the ASCII names of the fonts. For example:

```
fonts 4 R I B S
```

codeset *codesetName* — Code set for the particular printer or typesetter, where *CodesetName* is a valid code set name for use with the **iconv** command. The specified code set is used to define character entries in the charset section of font description files. For example:

```
codeset ISO8859-1
```

The **troff** command uses the specified *CodesetName* and the code set implied by the current locale to determine if code set conversions are necessary for the input characters. The **iconv** function is used to perform the code set conversion if necessary.

charset — Last keyword in the file 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 new line. 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 occurs only in multiples of two basic units and horizontal motion occurs only in one basic unit, **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 as follows:

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 0 (zero). Accepted ligatures are **ff fi fl ffi ffl**.

spacewidth *Number* — Specifies width of space if something other than the default (1/3 of an em space) is desired.

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
```

OR

```
Name "
```

where the value of the `Name` field is either a single-byte character or a special character name from the list found in the **DESC** file. The `Width` field 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, the `Code` field can be a multi-byte sequence.

For fonts of extended-character output devices, the `Code` field can be a multi-byte sequence that begins and ends with a double quotation mark. In the sequence, control or nonprinting characters can be represented by the following escape sequences:

`\n` — Produces a new line.

`\r` — Produces a return.

`\t` — Produces a tab.

`\b` — Produces a backspace.

`\"` — Produces a double quote.

`\xdd` — Produces a hexadecimal number, where `dd` is two hexadecimal digits.

`\ooo` — Produces 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 indicate that this name has the same values as the preceding line. The `Kerning` and `Code` fields are not used if the value of the `Width` field is a double quotation mark. 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

The **DESC.out** and *Font***.out** files were created as a result of executing the **makedev** program on the **DESC** file.

Prototype characters are provided for the charset section of the font table for fonts in large-character sets. Most characters in large-character sets, such as the Japanese, Chinese, and Korean character sets, have the same width. These prototype characters specify the width of characters with varying byte lengths. The kerning and code fields are not available for prototype character entries. These entries apply to all characters not explicitly defined in the charset section. It is assumed that the printer or typesetter code for characters handled through prototype characters is the same as the input code for the character after conversion by the **iconv** function. The following are the prototype character definitions:

X0	Width	Width of all characters that return a value of 0 for **csid()**.
X1	Width	Width of all 1-byte characters not defined elsewhere.
X1	Width	Width of all characters that return a value of 1 for **csid()**.
X2	Width	Width of all 2-byte characters not defined elsewhere.
X*i*	Width	Width of all characters that return a value of *i* for **csid()**.
X3	Width	Width of all 3-byte characters not defined elsewhere.
X4	Width	Width of all 4-byte characters not defined elsewhere.

For example, the following prototype character definitions apply to the Japanese character sets (both IBM-932 and IBM-eucJP):

```
X0   : alphanumeric characters
X1   : JIS level 1 and 2 Kanji characters in JISX0208.1990
X2   : Katakana characters
X3   : IBM selected characters
```

This **troff** font file is part of Formatting Tools in the Text Formatting System.

Files

/usr/lib/font/dev*Name***/DESC.out** file	Contains the description file for phototypesetter specified by the *Name* variable.
/usr/lib/font/dev*Name***/bitmaps** directory	Contains bitmap files.
/usr/lib/font/dev*Name*/*Font***.out** file	Contains the font description file for phototypesetter specified by the *Name* variable.

Related Information

The **troff** file format.

The **troff** command.

The **iconv** subroutine.

uconvdef Source File Format

Purpose

Defines UCS-2 (Unicode) conversion mappings for input to the **uconvdef** command.

Description

Conversion mapping values are defined using UCS-2 symbolic character names followed by character encoding (code point) values for the multibyte code set. For example,

```
<U0020>      \x20
```

represents the mapping between the `<U0020>` UCS-2 symbolic character name for the space character and the `\x20` hexadecimal code point for the space character in ASCII.

In addition to the code set mappings, directives are interpreted by the **uconvdef** command to produce the compiled table. These directives must precede the code set mapping section. They consist of the following keywords surrounded by < > (angle brackets), starting in column 1, followed by white space and the value to be assigned to the symbol:

<code_set_name> The name of the coded character set, enclosed in quotation marks (" "), for which the character set description file is defined.

<mb_cur_max> The maximum number of bytes in a multibyte character. The default value is 1.

<mb_cur_min> An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set. The value is less than or equal to **<mb_cur_max>**. If not specified, the minimum number is equal to **<mb_cur_max>**.

<escape_char> The escape character used to indicate that the character following is interpreted in a special way. This defaults to a backslash (\).

<comment_char> The character that, when placed in column 1 of a **charmap** line, is used to indicate that the line is ignored. The default character is the number sign (#).

<char_name_mask> A quoted string consisting of format specifiers for the UCS-2 symbolic names. This must be a value of AXXXX, indicating an alphabetic character followed by 4 hexadecimal digits. Also, the alphabetic character must be a U, and the hexadecimal digits must represent the UCS-2 code point for the character. An example of a symbolic character name based on this mask is `<U0020>` Unicode space character.

<uconv_class> Specifies the **type** of the code set. It must be one of the following:

SBCS	Single-byte encoding
DBCS	Stateless double-byte, single-byte, or mixed encodings
EBCDIC_STATEFUL	Stateful double-byte, single-byte, or mixed encodings
MBCS	Stateless multibyte encoding

This **type** is used to direct **uconvdef** on what type of table to build. It is also stored in the table to indicate the type of processing algorithm in the UCS conversion methods.

<locale> Specifies the default locale name to be used if locale information is needed.

<subchar> Specifies the encoding of the default substitute character in the multibyte code set.

The mapping definition section consists of a sequence of mapping definition lines preceded by a **CHARMAP** declaration and terminated by an **END CHARMAP** declaration. Empty lines and lines containing **<comment_char>** in the first column are ignored.

Symbolic character names in mapping lines must follow the pattern specified in the <**char_name_mask**>, except for the reserved symbolic name, <**unassigned**>, that indicates the associated code points are unassigned.

Each noncomment line of the character set mapping definition must be in one of the following formats:

1. `"%s %s %s/n", <symbolic-name>, <encoding>, <comments>`

   For example:

   ```
   <U3004>      \x81\x57
   ```

   This format defines a single symbolic character name and a corresponding encoding.

   The encoding part is expressed as one or more concatenated decimal, hexadecimal, or octal constants in the following formats:

   - `"%cd%d", <escape_char>, <decimal byte value>`
   - `"%cx%x", <escape_char> , <hexadecimal byte value>`
   - `"%c%o", <escape_char>, <octal byte value>`

   Decimal constants are represented by two or more decimal digits preceded by the escape character and the lowercase letter **d**, as in `\d97` or `\d143`. Hexadecimal constants are represented by two or more hexadecimal digits preceded by an escape character and the lowercase letter **x**, as in `\x61` or `\x8f`. Octal constants are represented by two or more octal digits preceded by an escape character.

   Each constant represents a single-byte value. When constants are concatenated for multibyte character values, the last value specifies the least significant octet and preceding constants specify successively more significant octets.

2. `"%s. . .%s %s %s/n", <symbolic-name>, <symbolic-name>, <encoding>, <comments>`

   For example:

   ```
   <U3003>...<U3006>   \x81\x56
   ```

   This format defines a range of symbolic character names and corresponding encodings. The range is interpreted as a series of symbolic names formed from the alphabetic prefix and all the values in the range defined by the numeric suffixes.

   The listed encoding value is assigned to the first symbolic name, and subsequent symbolic names in the range are assigned corresponding incremental values. For example, the line:

   ```
   <U3003>...<U3006>   \x81\x56
   ```

   is interpreted as:

   ```
   <U3003>      \x81\x56
   <U3004>      \x81\x57
   <U3005>      \x81\x58
   <U3006>      \x81\x59
   ```

3. `"<unassigned> %s. . .%s %s/n", <encoding>, <encoding>, <comments>`

This format defines a range of one or more unassigned encodings. For example, the line:

```
<unassigned>   \x9b...\x9c
```

is interpreted as:

```
<unassigned>   \x9b
<unassigned>   \x9c
```

Implementation Specifics

This command is part of Extended Commands in BOS Extensions 1.

Related Information

The **uconvdef** command.

Code Set Overview in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts*.

UIL File Format

Purpose

Contains information on the user interface for a widget-based application.

Description

User Interface Language (UIL) is used to describe the initial state of a user interface for a widget-based application. UIL describes the widgets used in the interface, the resources of those widgets, and the callbacks of those widgets. A UIL file is compiled into a user interface definition (UID) file using the **uil** command or the **Uil** callable compiler function. The contents of the compiled UID file can then be accessed by the various AIXwindows Resource Manager (MRM) functions from within an application program.

The syntax for the UIL is as follows:

MODULE *ModuleName*
[**NAMES = CASE_INSENSITIVE | CASE_SENSITIVE**]
[**CHARACTER_SET =** *CharacterSet*]
[**OBJECTS = {** *WidgetName* **= GADGET | WIDGET;** [...] **}**]
{ [
[*ValueSection*] |
[*ProcedureSection*] |
[*ListSection*] |
[*ObjectSection*] |
[*IdentifierSection*] |
[...]
] **}**
END MODULE;

File Format

UIL is a free-form language. This means that high-level constructs, such as object and value declarations, do not need to begin in any particular column and can span any number of lines. Low-level constructs, such as keywords and punctuation characters, can also begin in any column; however, except for string literals and comments, they cannot span lines.

The UIL compiler accepts input lines up to 132 characters in length.

MODULE *ModuleName*	The name by which the UIL module is known in the UID file. This name is stored in the UID file for later use in the retrieval of resources by the MRM. This module name is always uppercase.

NAMES = CASE_INSENSITIVE | CASE_SENSITIVE	Indicates whether names should be treated as case-sensitive or case-insensitive. The default is case-sensitive. The case-sensitivity clause should be the first clause in the module header and must precede any statement that contains a name. If names are case-sensitive in a UIL module, UIL keywords in that module must be in lowercase. Each name is stored in the UIL file in the same case as it appears in the UIL module. If names are case-insensitive, keywords can be in uppercase, lowercase, or mixed case, and the uppercase equivalent of each name is stored in the UID file.
CHARACTER_SET = *CharacterSet*	Specifies the default character set for string literals in the module that do not explicitly set their character set. In the absence of this clause, the default character set is the codeset component of the **LANG** environment variable, or the value of **XmFALLBACK_CHARSET** if **LANG** is not set or has no codeset component. The value of **XmFALLBACK_CHARSET** is defined by the UIL supplier, but is usually **ISO8859-1** (equivalent to **ISO_LATIN1**). Use of this clause turns off all localized string literal processing turned on by either the **-s** compiler flag or the **Uil_command_type** data structure element **use_setlocale_flag**.

OBJECTS = { *WidgetName* **= GADGET | WIDGET;}**	Indicates whether the widget or gadget form of the control specified by *WidgetName* variable is used by default. The widget form is used by default. The specified control should be one that has both a widget and gadget version, for example: **XmCascadeButton**, **XmLabel**, **XmPushButton**, **XmSeparator**, and **XmToggleButton**. The form of more than one control can be specified by delimiting them with `;` (semicolons). The gadget or widget form of an instance of a control can be specified with the **GADGET** and **WIDGET** keywords in a particular object declaration.
ValueSection	Provides a way to name a value expression or literal. The value name can then be referred to by declarations that occur elsewhere in the UIL module in any context where a value can be used. Values can be forward-referenced. See "Value Sections" for more detail.
ProcedureSection	Defines the callback functions used by a widget and the creation functions for user-defined widgets. These definitions are used for error checking. See "Procedure Sections" for more detail.
ListSection	Provides a way to group together a set of arguments, controls (children), callbacks, or procedures for later use in the UIL module. Lists can contain other lists so you can set up a hierarchy to clearly show which arguments, controls, callbacks, and procedures are common to which widgets. See "List Sections" for more detail.

ObjectSection	Defines the objects that make up the user interface of the application. You can reference the object names in declarations that occur elsewhere in the UIL module in any context where an object name can be used (for example, in a controls list, as a symbolic reference to a widget ID, or as the *TagValue* argument for a callback procedure). Objects can be forward-referenced. See "Object Sections" for more detail.
IdentifierSection	Defines a run-time binding of data to names that appear in the UIL module. See "Identifier Sections" for more detail.

The UIL file can also contain comments and include directives. These, as well as the main elements of the **UIL** file format, are described in the following sections.

Comments

Comments can take one of two forms, neither of which can be nested:

- The comment is introduced with the /* sequence followed by the text of the comment and terminated with the */ sequence. This form of comment can span multiple source lines.
- The comment is introduced with an ! (exclamation point) followed by the text of the comment and terminated by the end of the source line.

Value Sections

A value section consists of the **VALUE** keyword followed by a sequence of value declarations. It has the following syntax:

VALUE *ValueName* **:**
    [**EXPORTED** | **PRIVATE**] *ValueExpression* |
    IMPORTED *ValueType* **;**

ValueExpression is assigned to *ValueName*, or a *ValueType* is assigned to an imported value name. A value declaration provides a way to name a value expression or literal. The value name can be referred to by declarations that occur later in the UIL module in any context where a value can be used. Values can be forward-referenced.

EXPORTED A value that you define as exported is stored in the UID file as a named resource and can be referenced by name in other UID files. When you define a value as exported, MRM looks outside the module in which the exported value is declared to get its value at run time.

PRIVATE A private value is a value that is not imported or exported. A value that you define as private is not stored as a distinct resource in the UID file. You can reference a private value only in the UIL module containing the value declaration. The value or object is directly incorporated into anything in the UIL module that references the declaration.

IMPORTED A value that you define as imported is one that is defined as a named resource in a UID file. MRM resolves this declaration with the corresponding exported declaration at application run time.

By default, values and objects are private. The following is a list of the supported value types in UIL:

- **ANY**
- **ARGUMENT**
- **BOOLEAN**
- **COLOR**
- **COLOR_TABLE**
- **COMPOUND_STRING**
- **FLOAT**
- **FONT**
- **FONT_TABLE**
- **FONTSET**
- **ICON**
- **INTEGER**
- **INTEGER_TABLE**
- **KEYSYM**
- **REASON**
- **SINGLE_FLOAT**
- **STRING**
- **STRING_TABLE**
- **TRANSLATION_TABLE**
- **WIDE_CHARACTER**
- **WIDGET**

Procedure Sections

A procedure section consists of the **PROCEDURE** keyword followed by a sequence of procedure declarations. It has the following syntax:

PROCEDURE
    ProcedureName [**(** [*ValueType*] **)**] **;**

Use a procedure declaration to declare the following:

- A function that can be used as a callback function for a widget
- The creation function for a user-defined widget.

You can reference a procedure name in declarations that occur later in the UIL module in any context where a procedure can be used. Procedures can be forward-referenced. You *cannot* use a name that you used in another context as a procedure name.

In a procedure declaration, you have the option of specifying that a parameter is passed to the corresponding callback function at run time. This parameter is called the *callback tag*. You can specify the data type of the callback tag by putting the data type in parentheses following the procedure name. When you compile the module, the UIL compiler checks that the argument you specify in references to the procedure is of this type. Note that the data type of the callback tag must be one of the valid UIL data types. You can use a widget as a callback tag, as long as the widget is defined in the same widget hierarchy as the callback; that is, they must have a common ancestor that is in the same UIL hierarchy.

The following list summarizes how the UIL compiler checks argument type and argument count, depending on the procedure declaration:

No parameters	No argument type or argument count checking occurs. You can supply either 0 or 1 arguments in the procedure reference.
( )	Checks that the argument count is 0.
(ANY)	Checks that the argument count is 1. Does not check the argument type. Use the ANY data type to prevent type checking on procedure tags.
(*Type***)**	Checks for one argument of the specified type.
(*ClassName***)**	Checks for one widget argument of the specified widget class.

While it is possible to use any UIL data type to specify the type of a tag in a procedure declaration, you must be able to represent that data type in the programming language you are using. Some data types (such as integer, Boolean, and string) are common data types recognized by most programming languages. Other UIL data types (such as string tables) are more complicated and may require you to set up an appropriate corresponding data structure in the application in order to pass a tag of that type to a callback function.

You can also use a procedure declaration to specify the creation function for a user-defined widget. In this case, you specify no formal parameters. The procedure is called with the standard three arguments passed to all widget creation functions. See "AIXwindows Programming Overview" in *AIXwindows Programming Guide* for more information about widget creation functions.

List Sections

A list section consists of the **LIST** keyword followed by a sequence of list declarations. It has the following syntax:

LIST
    ListName **: {** *ListItem***;** [...] **}**

[...]

You can also use list sections to group together a set of arguments, controls (children), callbacks, or procedures for later use in the UIL module. Lists can contain other lists so you can set up a hierarchy to clearly show which arguments, controls, callbacks, and procedures are common to which widgets. You cannot mix the different types of lists; a list of a particular type cannot contain entries of a different list type or reference the name of a different list type. A list name is always private to the UIL module in which you declare the list and cannot be stored as a named resource in a UID file.

The additional list types are described in the following sections.

Arguments List Structure

An arguments list defines which arguments are specified in the arguments-list parameter when the creation function for a particular object is called at run time. An arguments list also specifies the values for those arguments. Arguments lists have the following syntax:

LIST *ListName* **: ARGUMENTS {**
ArgumentName = *ValueExpression***;**
[...] **}**
[...]

The argument name (*ArgumentName*) must be either a built-in argument name or a user-defined argument name that is specified with the **ARGUMENTS** function.

If you use a built-in argument name as an arguments list entry in an object definition, the UIL compiler checks the argument name to be sure that it is supported by the type of object that you are defining. If the same argument name is displayed more than once in a given arguments list, the last entry that uses that argument name supersedes all previous entries with that name, and the compiler issues a message.

Some arguments, such as **XmNitems** and **XmNitemCount**, are coupled by the UIL compiler. When you specify one of the coupled arguments, the compiler also sets the other one. The coupled argument is not available to you.

AIXwindows and the X Toolkit (Intrinsics) support *constraint arguments*. A constraint argument is one that is passed to children of an object, beyond those arguments normally available. For example, the **Form** widget grants a set of constraint arguments to its children. These arguments control the position of the children within the **Form** widget.

Unlike the arguments used to define the attributes of a particular widget, constraint arguments are used exclusively to define additional attributes of the children of a particular widget. These attributes affect the behavior of the children within their parent. To supply constraint arguments to the children, include the arguments in the arguments list for the child.

Callbacks List Structure

Use a callbacks list to define which callback reasons are to be processed by a particular widget at run time. Callback lists have the following syntax:

LIST
ListName **: CALLBACKS {**

ReasonName **=** **PROCEDURE** *ProcedureName* [**(** [*ValueExpression*] **)**]**;** |
ReasonName = *ProcedureList* **;**
[...] **}**
[...]

For AIXwindows widgets, the reason name must be a built-in reason name. For a user-defined widget, you can use a reason name that you previously specified using the **REASON** function. If you use a built-in reason in an object definition, the UIL compiler ensures that reason is supported by the type of object you are defining.

If the same reason is displayed more than once in a callbacks list, the last entry referring to that name supersedes all previous entries using the same reason. The UIL compiler then issues a diagnostic message.

If you specify a named value for the procedure argument (callback tag), the data type of the value must match the type specified for the callback tag in the corresponding procedure declaration. When specifying a widget name as a procedure value expression, you must also specify the type of the widget and a space before the name of the widget.

Because the UIL compiler produces a UID file rather that an object module (**.o**), the binding of the UIL name to the address of the entry point and then to the procedure is not done by the loader. Instead, this binding is established at run time with the **MrmRegisterNames** MRM function. You call this function before fetching any objects, giving it both the UIL names and the procedure addresses of each callback. The name you register with MRM in the application program must match the name you specified for the procedure in the UIL module.

Each callback procedure received three arguments. The first two arguments have the same form for each callback. The form of the third argument varies from object to object.

The first argument is the address of the data structure maintained by the AIXwindows for this object instance. This address is called the widget ID for this object.

The second argument is the address of the value you specified in the callbacks list for this procedure. If you do not specify an argument, the address is null.

The third argument is the reason name you specified in the callbacks list.

Controls List Structure

A controls list defines which objects are children of, or controlled by, a particular object. Each entry in a controls list has the following syntax:

LIST
ListName **: CONTROLS {**
[*ChildName*] [**MANAGED** | **UNMANAGED**] *ObjectDefinition***;**
[...] **}**
[...]

If you specify the **MANAGED** keyword at run time, the object is created and managed; if you specify the **UNMANAGED** keyword at run time, the object is only created. Objects are managed by default.

You can use the *ChildName* parameter to specify resources for the automatically created children of a particular control. Names for automatically created children are formed by appending **Xm_** to the name of the child widget. This name is specified in the documentation for the parent widget.

Unlike the arguments list and the callbacks list, a controls list entry that is identical to a previous entry does *not* supersede the previous entry. At run time, each controls list entry causes a child to be created when the parent is created. If the same object definition is used for multiple children, multiple instances of the child are created at run time.

Procedures List Structure

You can specify multiple procedures for a callback reason in UIL by defining a procedures list. Just as with other list types, procedures lists can be defined in-line or in a list section and referenced by name.

If you define a reason more than once (for example, when the reason is defined both in a referenced procedures list and in the callbacks list for the object), previous definitions are overridden by the latest definition. The syntax for a procedures list is as follows:

LIST
        ListName **: PROCEDURES {**
                ProcedureName [**(** [*ValueExpression*] **)**]**;**
                [...] **}**
        [...]

When specifying a widget name as a procedure value expression, you must also specify the type of the widget and a space before the name of the widget.

Object Sections

An object section consists of the **OBJECT** keyword followed by a sequence of object declarations. It has the following syntax:

OBJECT *ObjectName* **:**
        [**EXPORTED** | **PRIVATE** | **IMPORTED**] *ObjectType*
                [**PROCEDURE** *CreationFunction*]
                [*ObjectName* [**WIDGET** | **GADGET**] | **{** *ListDefinitions* **}**]

Use an object declaration to define the objects that are stored in the UID file. You can reference the object name in declarations that occur elsewhere in the UIL module in any context where an object name can be used (for example, in a controls list, as a symbolic reference to a widget ID, or as the *TagValue* argument for a callback procedure). Objects can be *forward-referenced*, meaning that you can declare an object name after you have referenced it. All references to an object name must be consistent with the type of the object, as specified in the object declaration. You can specify an object as exported, imported, or private.

The object definition can contain a sequence of lists that define the arguments, hierarchy, and callbacks for the widget. You can only specify one list of each type for an object. When you declare a user-defined widget, you must include a reference to the widget creation function for the user-defined widget.

Use the **GADGET** or **WIDGET** keyword to specify the object type or to override the default variant for this object type. You can use the AIXwindows name of an object type that has a gadget variant (for example, **XmLabelGadget**) as an attribute of an object declaration. The *ObjectType* can be any object type, including gadgets. You need to specify the **GADGET** or **WIDGET** keyword only in the declaration of an object, not when you reference the object. You *cannot* specify the **GADGET** or **WIDGET** keyword for a user-defined object; user-defined objects are always widgets.

Identifier Sections

The identifier section allows you to define an *identifier*, a mechanism that achieves run-time binding of data to names that appear in a UIL module. The identifier section consists of the reserved **IDENTIFIER** keyword, followed by a list of names. Each name is followed by a semicolon (;). The syntax is as follows:

IDENTIFIER *IdentifierName***;** [...**;**]

You can use these names later in the UIL module as either the value of an argument to a widget or the tag value to a callback procedure. At run time, use the **MrmRegisterNames** and **MrmRegisterNamesInHierarchy** MRM functions to bind the identifier name with the data (or, in the case of callbacks, with the address of the data) associated with the identifier.

Each UIL module has a single name space; therefore, you cannot use the name you used for a value, object, or procedure as an identifier name in the same module.

The UIL compiler does not do any type checking on the use of identifiers in a UIL module. Unlike a UIL value, an identifier does not have a UIL type associated with it. Regardless of what particular type a widget argument or callback procedure tag is defined to be, you can use an identifier in that context instead of a value of the corresponding type.

To reference these identifier names in a UIL module, use the name of the identifier wherever you want its value to be used.

Include Directives

The include directive incorporates the contents of a specified file into a UIL module. This mechanism allows several UIL modules to share common definitions. The syntax for the include directive is as follows:

INCLUDE FILE *FileName* **;**

The UIL compiler replaces the include directive with the contents of the include file and processes it as if these contents were displayed in the current UIL source file.

You can nest include files, meaning that an include file can contain include directives. The UIL compiler can process up to 100 references (including the file containing the UIL module). Therefore, you can include up to 99 files in a single UIL module, including nested files. Each time a file is opened counts as a reference; therefore, including the same file twice counts as two references.

The character expression is a file specification that identifies the file to be included. The rules for finding the specified file are similar to the rules for finding header, or **.h**, files using the include directive, **#include**, with a quoted string in C language. The **uil** command uses the **-I** option for specifying a search directory for include files. Search rules are as follows:

- If you supply a directory, the UIL compiler searches only that directory for the include file.
- If you do not supply a directory, the UIL compiler searches for the include file in the directory of the main source file.
- If the include file is not found in the main source file directory, the compiler looks in the same directory as the source file.

Language Syntax

This section contains information on the following:

- Names and Strings
- Data Types
- String Literals
- Integer Literals
- Boolean Literals
- Floating-Point Literals
- ANY Data Type
- Expressions
- Functions.

Names and Strings

Names can consist of any of the characters A to Z, a to z, 0 to 9, $ (dollar sign), and _ (underscore). Names cannot begin with a digit (0 to 9). The maximum length of a name is 31 characters.

UIL gives you a choice of either case-sensitive or case-insensitive names through a clause in the **MODULE** header. For example, if names are case-sensitive, the names "sample" and "Sample" are distinct from each other. If names are case-insensitive, these names are treated as the same name and can be used interchangeably. By default, UIL assumes names are case-sensitive.

In case-insensitive mode, the compiler outputs all names in the UID file in uppercase form. In case-sensitive mode, names are displayed in the UIL file exactly as they are displayed in the source file.

The following lists the reserved keywords, which *cannot* be used for programmer-defined names:

ARGUMENTS	**CALLBACKS**	**CONTROLS**	**END**
EXPORTED	**FALSE**	**GADGET**	**IDENTIFIER**
INCLUDE	**LIST**	**MODULE**	**OFF**
ON	**OBJECT**	**PRIVATE**	**PROCEDURE**
PROCEDURES	**TRUE**	**VALUE**	**WIDGET**

The following lists UIL unreserved keywords. These keywords can be used as programmer-defined names; however, if you use any of these keywords as names, you cannot use the UIL-supplied form of that keyword.

Built-in argument names (for example, **XmNx**, **XmNheight**)
Built-in reason names (for example, **XmNactivateCallback**, **XmNhelpCallback**)
Character set names (for example, **ISO_LATIN1**, **ISO_HEBREW_LR**)
Constant value names (for example, **XmMENU_OPTION**, **XmBROWSE_SELECT**)
Object types (for example, **XmPushButton**, **XmBulletinBoard**)

ANY	**FILE**	**IMPORTED**
ARGUMENT	**FLOAT**	**REASON**
ASCIZ_STRING_TABLE	**FONT**	**RGB**
ASCIZ_TABLE	**FONTSET**	**SINGLE_FLOAT**
BACKGROUND	**FONT_TABLE**	**STRING**
BOOLEAN	**FOREGROUND**	**STRING_TABLE**
CASE_INSENSITIVE	**ICON**	**TRANSLATION_TABLE**
CASE_SENSITIVE	**INTEGER**	**UNMANAGED**
CHARACTER_SET	**INTEGER_TABLE**	**USER_DEFINED**
COLOR	**KEYSYM**	**VERSION**
COLOR_TABLE	**MANAGED**	**WIDE_CHARACTER**
COMPOUND_STRING	**NAMES**	**WIDGET**
COMPOUND_STRING_TABLE	**OBJECTS**	**XBITMAPFILE**
	RIGHT_TO_LEFT	

String literals can be composed of uppercase and lowercase letters, digits, and punctuation characters. Spaces, tabs, and comments are special elements in the language. They are a means of delimiting other elements, such as two names. One or more of these elements can be displayed before or after any other element in the language. However, spaces, tabs, and comments that are displayed in string literals are treated as character sequences rather than delimiters.

Data Types

UIL provides literals for several of the value types it supports. Some of the value types are not supported as literals (for example, pixmaps and string tables). You can specify values for these types by using functions described in the "Functions" section . UIL directly supports the following literal types:

- String literal
- Integer literal
- Boolean literal
- Floating-point literal

UIL also includes the **ANY** data type, which is used to turn off compile-time checking of data types.

String Literals

A string literal is a sequence of 0 or more 8-bit or 16-bit characters or a combination delimited by ' (single quotation marks) or " (double quotation marks). String literals can also contain multibyte characters delimited with double quotation marks. String literals can be no more than 2,000 characters long.

A single-quoted string literal can span multiple source lines. To continue a single-quoted string literal, end the continued line with a \ (backslash). The literal continues with the first character on the next line.

Double-quoted string literals cannot span multiple source lines. (Because double-quoted strings can contain escape sequences and other special characters, you cannot use the backslash character to designate the continuation of the string.) To build a string value that must span multiple source lines, use the concatenation operation that is described later in this section.

The syntax of a string literal can be one of the following:

```
'[CharacterString]'
[#CharSet]"[CharacterString]"
```

Both string forms associate a character set with a string value. UIL uses the following rules to determine the character set and storage format for string literals:

- A string declared as **'***String***'** is equivalent to #*CurCharSet***"***String***"**, where *CurCharSet* is the codeset portion of the value of the **LANG** environment variable. If the **LANG** environment variable is not set or has no code set component, *CurCharSet* is the value of **XmFALLBACK_CHARSET**. By default, **XmFALLBACK_CHARSET** is **ISO8859-1** (equivalent to **ISO_LATIN1**), but vendors can define a different default.
- A string declared as **"***String***"** is equivalent to #*CharSet***"***String***"** if you specified *CharSet* as the default character set for the module. If no default character set has been specified for the module and either the **-s** option is provided to the **uil** command or the **use_setlocale_flag** value is set for the **Uil** function callable compiler, the string is interpreted to be a string in the current locale. This means that the string is parsed in the locale of the user by calling **setlocale** and its character set is set to a value of **XmFONTLIST_DEFAULT_TAG**. If the string is converted to a compound string, it is stored as a locale-encoded text segment. Otherwise, **"***String***"** is equivalent to #*CurCharSet***"***String***"**, where *CurCharSet* is interpreted as described for single-quoted strings.
- A string of the form **"***String***"** or #*CharSet***"***String***"** is stored as a null-terminated string.

The following lists the character sets supported by the UIL compiler for string literals. Note that several UIL names map to the same character set. In some cases, the UIL name influences how string literals are read. For example, strings identified by a UIL character set name ending in **_LR** are read left-to-right. Names that end in a different number reflect different fonts (for example, **ISO_LATIN1** or **ISO_LATIN6**). All character sets in this list are represented by 8 bits.

UIL Name	Description
ISO_LATIN1	GL: ASCII, GR: Latin-1 Supplement
ISO_LATIN2	GL: ASCII, GR: Latin-2 Supplement
ISO_ARABIC	GL: ASCII, GR: Latin-Arabic Supplement
ISO_LATIN6	GL: ASCII, GR: Latin-Arabic Supplement
ISO_GREEK	GL: ASCII, GR: Latin-Greek Supplement
ISO_LATIN7	GL: ASCII, GR: Latin-Greek Supplement
ISO_HEBREW	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_LATIN8	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_HEBREW_LR	GL: ASCII, GR: Latin-Hebrew Supplement
ISO_LATIN8_LR	GL: ASCII, GR: Latin-Hebrew Supplement
JIS_KATAKANA	GL: JIS Roman, GR: JIS Katakana

Following are the parsing rules for each of the character sets:

All character sets	Character codes in the range 00 to 1F, 7F, and 80 to 9F are control characters including both bytes of 16-bit characters. The compiler flags these as illegal characters.
ISO_LATIN1, **ISO_LATIN2**, **ISO_ARABIC**, **ISO_LATIN6**, **ISO_GREEK**, **ISO_LATIN7**	These sets are parsed from left to right. The escape sequences for null-terminated strings are also supported by these character sets.
ISO_HEBREW, **ISO_LATIN8**	These sets are parsed from right to left. For example, the string `#ISO_HEBREW"012345"` generates a primitive string `"543210"` with the character set **ISO_HEBREW**. A DDIS descriptor for such a string has this segment marked as being right to left. The escape sequences for null-terminated strings are also supported by these character sets, and the characters that compose the escape sequences are in left-to-right order. For example, you type `\n`, not `n\`.
ISO_HEBREW_LR, **ISO_LATIN8_LR**	These sets are parsed from left to right. For example, the string `#ISO_HEBREW"012345"` generates a primitive string `"012345"` with the character set **ISO_HEBREW**. A DDIS descriptor for such a string marks this segment as being left to right. The escape sequences for null-terminated strings are also supported by these character sets.
JIS_KATAKANA	This set is parsed from left to right. The escape sequences for null-terminated strings are also supported by these character sets. Note that the \ (backslash) can be displayed as a yen symbol.

In addition to designating parsing rules for strings, character set information remains an attribute of a compound string. If the string is included in a string consisting of several concatenated segments, the character set information is included with that string segment. This gives AIXwindows the information it needs to decipher the compound string and choose a font to display the string.

For an application interface displayed only in English, UIL lets you ignore the distinctions between the two uses of strings. The compiler recognizes by context when a string must be passed as a null-terminated string or as a compound string.

The UIL compiler recognizes enough information about the various character sets to correctly parse string literals. The compiler also issues errors if you use a compound string in a context that supports only null-terminated strings.

Since the character set names are keywords, you must put them in lowercase if case-sensitive names are in force. If names are case-insensitive, character set names can be uppercase, lowercase, or mixed case.

In addition to the built-in character sets recognized by UIL, you can define your own character sets with the **CHARACTER_SET** function. You can use the **CHARACTER_SET** function anywhere a character set can be specified.

String literals can contain characters with the eighth (high-order) bit set. You cannot type control characters (00 to 1F, 7F, and 80 to 9F) directly in a single-quoted string literal. However, you can represent these characters with escape sequences. The following list shows the escape sequences for special characters:

\b	Backspace
\f	Form-feed
\n	New-line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\’	Single quotation mark
\"	Double quotation mark
\\	Backslash
Integer\	Character whose internal representation is given by *Integer* (in the range 0 to 255 decimal).

Note: Escape sequences are processed literally in strings that are parsed in the current locale (localized strings).

The UIL compiler does not process new-line characters in compound strings. The effect of a new-line character in a compound string depends only on the character set of the string. The result is not guaranteed to be a multiline string.

Compound String Literals

A compound string consists of a string of 8-bit, 16-bit, or multibyte characters, a named character set, and a writing direction. Its UIL data type is **compound_string**.

The writing direction of a compound string is implied by the character set specified for the string. You can explicitly set the writing direction for a compound string by using the **COMPOUND_STRING** function.

A compound string can consist of a sequence of concatenated compound strings, null-terminated strings, or a combination of both, each of which can have a different character set property and writing direction. Use the & (ampersand) concatenation operator to create a sequence of compound strings.

Each string in the sequence is stored, including the character set and writing direction information.

Generally, a string literal is stored in the UID file as a compound string when the literal consists of concatenated strings having different character sets or writing directions, or when you use the string to specify a value for an argument that requires a compound string value. If you want to guarantee that a string literal is stored as a compound string, you *must* use the **COMPOUND_STRING** function.

Data Storage Consumption for String Literals

The way a string literal is stored in the UID file depends on how you declare and use the string. The UIL compiler automatically converts a null-terminated string to a compound string if you use the string to specify the value of an argument that requires a compound string. However, this conversion is costly in terms of storage consumption.

PRIVATE, **EXPORTED**, and **IMPORTED** string literals require storage for a single allocation when the literal is declared; thereafter, storage is required for each reference to the literal. Literals declared in-line require storage for both an allocation and a reference.

The following list summarizes data storage consumption for string literals. The storage requirement for an allocation consists of a fixed portion and a variable portion. The fixed portion of an allocation is roughly the same as the storage requirement for a reference (a few bytes). The storage consumed by the variable portion depends on the size of the literal value (the length of the string). To conserve storage space, avoid making string literal declarations that result in an allocation per use.

Declaration	Data Type	Used As	Storage Requirements Per Use
In-line	Null-terminated	Null-terminated	An allocation and a reference (within the module)
Private	Null-terminated	Null-terminated	A reference (within the module)
Exported	Null-terminated	Null-terminated	A reference (within the UID hierarchy)
Imported	Null-terminated	Null-terminated	A reference (within the UID hierarchy)
In-line	Null-terminated	Compound	An allocation and a reference (within the module)
Private	Null-terminated	Compound	An allocation and a reference (within the module)
Exported	Null-terminated	Compound	A reference (within the UID hierarchy)
Imported	Null-terminated	Compound	A reference (within the UID hierarchy
In-line	Compound	Compound	An allocation and a reference (within the module)
Private	Compound	Compound	A reference (within the module)
Exported	Compound	Compound	A reference (within the UID hierarchy)
Imported	Compound	Compound	A reference (within the UID hierarchy)

Integer Literals

An integer literal represents the value of a whole number. Integer literals have the form of an optional sign followed by one or more decimal digits. An integer literal must not contain embedded spaces or commas.

Integer literals are stored in the UID file as long integers. Exported and imported integer literals require a single allocation when the literal is declared; thereafter, a few bytes of storage are required for each reference to the literal. Private integer literals and those declared in-line require allocation and reference storage per use. To conserve storage space, avoid making integer literal declarations that result in an allocation per use.

The following list shows data storage consumption for integer literals:

Declaration	Storage Requirements Per Use
In-line	An allocation and a reference (within the module).
Private	An allocation and a reference (within the module).
Exported	A reference (within the UID hierarchy).
Imported	A reference (within the UID hierarchy).

Boolean Literals

A Boolean literal represents the True value (reserved keyword **TRUE** or **On**) or False value (reserved keyword **FALSE** or **Off**). These keywords are subject to case-sensitivity rules.

In a UID file, **TRUE** is represented by the integer value 1 and **FALSE** is represented by the integer value 0.

Data storage consumption for Boolean literals is the same as that for integer literals.

Floating-Point Literals

A floating-point literal represents the value of a real (or float) number. Floating-point literals have the following form:

```
[+|-][Integer].Integer[E|e[+|-]Exponent]
```

For maximum portability, a floating-point literal can represent values in the range 1.0E-37 to 1.0E+37 with at least six significant digits. On many machines, this range is wider, with more significant digits. A floating-point literal must not contain embedded spaces or commas.

Floating-point literals are stored in the UID file as double-precision, floating-point numbers. The following gives examples of valid and invalid floating-point notation for the UIL compiler:

Valid Floating-Point Literals	Invalid Floating-Point Literals
1.0	1e1 (no decimal point)
.1	E-1 (no decimal point or digits)
3.1415E-2 (equals .031415)	2.87 e6 (embedded blanks)
-6.29e7 (equals -62900000)	2.0e100 (out of range)

Data storage consumption for floating-point literals is the same as that for integer literals.

ANY Data Type

The purpose of the **ANY** data type is to shut off the data-type checking feature of the UIL compiler. You can use the **ANY** data type for either of the following:

- Specifying the type of a callback procedure tag.
- Specifying the type of a user-defined argument.

You can use the **ANY** data type when you need to use a type not supported by the UIL compiler or when you want the data-type restrictions imposed by the compiler to be relaxed. For example, you might want to define a widget having an argument that can accept different types of values, depending on run-time circumstances.

If you specify that an argument takes an **ANY** value, the compiler does not check the type of the value specified for that argument. Therefore, you need to take care when specifying a value for an argument of the **ANY** data type. You may get unexpected results at run time if you pass a value having a data type that the widget does not support for that argument.

Expressions

UIL includes compile-time value expressions. These expressions can contain references to other UIL values, but cannot be forward-referenced.

The following lists the set of operators in UIL that allow you to create integer, real, and Boolean values based on other values defined with the UIL module. In the list, a precedence of 1 is the highest.

Operator	Operand Types	Meaning	Precedence
~	Boolean	NOT	1
	integer	Ones complement	
-	float	Negate	1
	integer	Negate	
+	float	NOP	1
	integer	NOP	
*****	float,float	Multiply	2
	integer,integer	Multiply	
/	float,float	Divide	2
	integer,integer	Divide	
+	float,float	Add	3
	integer,integer	Add	
-	float,float	Subtract	3
	integer,integer	Subtract	
>>	integer,integer	Shift right	4
<<	integer,integer	Shift left	4
&	Boolean,Boolean	AND	5
	integer,integer	Bitwise AND	
	string,string	Concatenate	
|	Boolean,Boolean	OR	6
	integer,integer	Bitwise OR	
^	Boolean,Boolean	XOR	6
	integer,integer	Bitwise XOR	

A string can be either a single compound string or a sequence of compound strings. If the two concatenated strings have different properties (such as writing direction or character set), the result of the concatenation is a multisegment compound string.

The string resulting from the concatenation is a null-terminated string unless one or more of the following conditions exists:

- One of the operands is a compound string.
- The operands have different character set properties.
- The operands have different writing directions.

If one or more of previous conditions are met, the resulting string is a compound string. You cannot use imported or exported values as operands of the concatenation operator.

The result of each operator has the same type as its operands. You cannot mix types in an expression without using conversion functions.

You can use parentheses to override the normal precedence of operators. In a sequence of unary operators, the operations are performed in right-to-left order. For example, `- + -A` is equivalent to `-(+(-A))`. In a sequence of binary operators of the same precedence, the operations are performed in left-to-right order. For example, `A*B/C*D` is equivalent to `((A*B)/c)*D`.

A value declaration gives a value a name. You cannot redefine the value of that name in a subsequent value declaration. You can use a value containing operators and functions anywhere you can use a value in a UIL module. You cannot use imported values as operands in expressions.

Several of the binary operators are defined for multiple data types. For example, the operator for multiplication (*) is defined for both floating-point and integer operands.

For the UIL compiler to perform these binary operations, both operands must be of the same type. If you supply operands of different data types, the UIL compiler automatically converts one of the operands to the type of the other according to the following conversion rules:

- If the operands are an integer and a Boolean, the Boolean is converted to an integer.
- If the operands are an integer and a floating-point, the integer is converted to a floating-point.
- If the operands are a floating-point and a Boolean, the Boolean is converted to a floating-point.

You can also explicitly convert the data type of a value by using one of the **INTEGER**, **FLOAT**, or **SINGLE_FLOAT** conversion functions.

Functions

UIL provides functions to generate the following types of values:

- Character sets
- Keysyms
- Colors
- Pixmaps
- Single-precision, floating-point numbers
- Double-precision, floating-point numbers
- Fonts
- Font sets

- Font tables
- Compound strings
- Compound string tables
- ASCIZ (null-terminated) string tables
- Wide character strings
- Widget class names
- Integer tables
- Arguments
- Reasons
- Translation tables.

All examples in the following sections assume case-insensitive mode. Keywords are shown in uppercase letters to distinguish them from user-specified names, which are shown in mixed-case italics. This use of uppercase letters is not required in case-insensitive mode. In case-sensitive mode, keywords must be in lowercase letters.

CHARACTER_SET(*StringExpression*[**,***Property*[**,** ...]]**)**

You can define your own character sets with the **CHARACTER_SET** function. You can use the **CHARACTER_SET** function anywhere a character set can be specified.

The result of the **CHARACTER_SET** function is a character set with the name *StringExpression* and the properties you specify. *StringExpression* must be a null-terminated string. You can optionally include one or both of the following clauses to specify properties for the resulting character set:

```
RIGHT_TO_LEFT = BooleanExpression
SIXTEEN_BIT = BooleanExpression
```

The **RIGHT_TO_LEFT** clause sets the default writing direction of the string from right to left if *BooleanExpression* is True, and left to right otherwise.

The **SIXTEEN_BIT** clause allows the strings associated with this character set to be interpreted as 16-bit characters if *BooleanExpression* is True, and 8-bit characters otherwise.

KEYSYM(*StringLiteral***)**

The **KEYSYM** function is used to specify a keysym for a mnemonic resource. The *StringLiteral* must contain exactly one character. If the **-s** compiler flag is used, *StringLiteral* which uses double quotes must specify a character set.

COLOR(*StringExpression*[**,FOREGROUND**|**BACKGROUND**]**)**

The **COLOR** function supports the definition of colors. Using the **COLOR** function, you can designate a value to specify a color and use that value for arguments requiring a color value. The string expression names the color you want to define. The optional **FOREGROUND** and **BACKGROUND** keywords identify how the color is to be displayed on a monochrome device when the color is used in the definition of a color table.

The UIL compiler does not have built-in color names. Colors are a server-dependent attribute of an object. Colors are defined on each server and may have different red-green-blue (RGB) values on each server. The string you specify as the color argument must be recognized by the server on which your application runs.

In a UID file, UIL represents a color as a character string. MRM calls X translation functions that convert a color string to the device-specific pixel value. If you are running on a monochrome server, all colors translate to black or white. If you are on a color server, the color names translate to their proper colors if the following conditions are met:

- The color is defined.
- The color map is not yet full.

If the color map is full, even valid colors translate to black or white (foreground or background).

Generally, interfaces do not specify colors for widgets. This enables the selection of colors to be controlled by the user through the **.Xdefaults** file.

To write an application that runs on both monochrome and color devices, you need to specify which colors in a color table (defined with the **COLOR_TABLE** function) map to the background and which colors map to the foreground. UIL lets you use the **COLOR** function to map the color red to the background color on a monochrome device as follows:

```
VALUE c: COLOR ( 'red',BACKGROUND );
```

Mapping is necessary only when the MRM is given a color and the application is to be displayed on a monochrome device. In this case, each color is considered to be in one of the following three categories:

- The color is mapped to the background color on the monochrome device.
- The color is mapped to the foreground color on the monochrome device.
- Monochrome mapping is undefined for this color.

If the color is mapped to the foreground or background color, MRM substitutes the foreground or background color, respectively. If you do not specify the monochrome mapping for a color, MRM passes the color string to AIXwindows for mapping to the foreground or background color.

RGB(*RedInteger***,** *GreenInteger***,** *BlueInteger***)**

The three integers define the values for the red, green, and blue components of the color, in that order. The values of these components can range from 0 to 65,535, inclusive.

In a UID file, UIL represents an RGB value as three integers. MRM calls X translation functions that convert the integers to the device-specific pixel value. If you are running on a monochrome server, all colors translate to black or white. If you are on a color server, RGB values translate to their proper colors if the color map is not yet full. If the color map is full, values translate to black or white (foreground or background).

COLOR_TABLE(*ColorExpression*=**'***Character***'**[**,** ...]**)**

The color expression is a previously defined color, a color defined in-line with the **COLOR** function, or the phrase **BACKGROUND COLOR** or **FOREGROUND COLOR**. The character can be any valid UIL character.

The **COLOR_TABLE** function provides a device-independent way to specify a set of colors. The **COLOR_TABLE** function accepts either previously defined UIL color names or in-line color definitions (using the **COLOR** function). A color table must be private because its contents must be known by the UIL compiler to construct an icon. The colors within a color table, however, can be imported, exported, or private.

The single letter associated with each color is the character you use to represent that color when creating an icon. Each letter used to represent a color must be unique within the color table.

ICON([**COLOR_TABLE**=*ColorTableName***,**] *Row*[**,** ...]**)**

The color table name must refer to a previously defined color table. The row is a character expression that gives one row of the icon.

The **ICON** function describes a rectangular icon that is *x* pixels wide and *y* pixels high. The strings surrounded by single quotation marks describe the icon. Each string represents a row in the icon; each character in the string represents a pixel.

The first row in an icon definition determines the width of the icon. All rows must have the same number of characters as the first row. The height of the icon is dictated by the number of rows.

The first argument of the **ICON** function (the color table specification) is optional and identifies the colors that are available in this icon. By using the single letter associated with each color, you can specify the color of each pixel in the icon. The icon must be constructed of characters defined in the specified color table.

A default color table is used if you omit the argument specifying the color table. To make use of the default color table, the rows of your icon must contain only spaces and asterisks. The default color table is defined as follows:

```
COLOR_TABLE( BACKGROUND COLOR = ' ', FOREGROUND COLOR = '*' )
```

You can define other characters to represent the background color and foreground color by replacing the space and asterisk in the **BACKGROUND COLOR** and **FOREGROUND COLOR** clauses shown in the example statement. You can specify icons as private, imported, or exported. Use the **MrmFetchIconLiteral** MRM function to retrieve an exported icon at run time.

XBITMAPFILE(*StringExpression***)**

The **XBITMAPFILE** function is similar to the **ICON** function in that both describe a rectangular icon that is *x* pixels wide and *y* pixels high. However, the **XBITMAPFILE** function allows you to specify an external file containing the definition of an X bitmap, while all **ICON** function definitions must be coded directly within UIL. X bitmap files can be generated by many different X applications. UIL reads these files through the **XBITMAPFILE** function, but does not support creation of these files. The X bitmap file specified as the argument to the **XBITMAPFILE** function is read by MRM at application run time.

The **XBITMAPFILE** function returns a value of type pixmap and can be used anywhere a pixmap data type is expected.

SINGLE_FLOAT(*RealNumberLiteral***)**

The **SINGLE_FLOAT** function lets you store floating-point literals in UIL files as single-precision, floating-point numbers. Single-precision, floating-point numbers can often be stored using less memory than double-precision, floating-point numbers. The *RealNumberLiteral* can be either an integer literal or a floating-point literal. A value defined using this function cannot be used in an arithmetic expression.

FLOAT(*RealNumberLiteral***)**

The **FLOAT** function lets you store floating-point literals in UIL files as double-precision, floating-point numbers. The *RealNumberLiteral* can be either an integer literal or a floating-point literal.

FONT(*StringExpression*[**,CHARACTER_SET**=*CharSet*]**)**

You define fonts with the **FONT** function. Using the **FONT** function, you designate a value to specify a font and use that value for arguments that require a font value. The UIL compiler has no built-in fonts.

Each font makes sense only in the context of a character set. The **FONT** function has an additional parameter to let you specify the character set for the font. This parameter is optional; if you omit it, the default character set depends on the value of the **LANG** environment variable. If **LANG** is not set, the default character set is set to **XmFALLBACK_CHARSET**.

The string expression specifies the name of the font and the clause **CHARACTER_SET**=*CharSet* specifies the character set for the font. The string expression used in the **FONT** function cannot be a compound string.

FONTSET(*StringExpression*[**,**...][**,CHARACTER_SET=***CharSet*]**)**

You define fontsets with the **FONTSET** function. Using the **FONTSET** function, you designate a set of values to specify a font and use those values for arguments that require a fontset value. The UIL compiler has no built-in fonts.

Each font makes sense only in the context of a character set. The **FONTSET** function has an additional parameter to let you specify the character set for the font. This parameter is optional; if you omit it, the default character set depends on the value of the **LANG** environment variable. If **LANG** is not set, the default character set is set to **XmFALLBACK_CHARSET**.

The string expression specifies the name of the font and the clause **CHARACTER_SET=***CharSet* specifies the character set for the font. The string expression used in the **FONTSET** function cannot be a compound string.

FONT_TABLE(*FontExpression*[**,**...]**)**

A font table is a sequence of pairs of fonts and character sets. At run time when an object needs to display a string, the object scans the font table for the character set that matches the character set of the string to be displayed. UIL provides the **FONT_TABLE** function to let you supply such an argument. The font expression is created with the **FONT** and **FONTSET** functions.

If you specify a single font value to specify an argument that requires a font table, the UIL compiler automatically converts a font value to a font table.

COMPOUND_STRING(*StringExpression*[**,***Property*[**,**...]]**)**

Use the **COMPOUND_STRING** function to set properties of a null-terminated string and to convert it into a compound string. The properties you can set are the character set, writing direction, and separator.

The result of the **COMPOUND_STRING** function is a compound string with the string expression as its value. You can optionally include one or more of the following clauses to specify properties for the resulting compound string:

CHARACTER_SET=*CharacterSet*
RIGHT_TO_LEFT=*BooleanExpression*
SEPARATE=*BooleanExpression*

The **CHARACTER_SET** clause specifies the character set for the string. If you omit the **CHARACTER_SET** clause, the resulting string has the same character set as *StringExpression*.

The **RIGHT_TO_LEFT** clause sets the writing direction of the string from right to left if *BooleanExpression* is True. Otherwise, writing direction is left to right. Specifying this argument does not cause the value of the string expression to change. If you omit the **RIGHT_TO_LEFT** argument, the resulting string has the same writing direction as *StringExpression*.

The **SEPARATE** clause appends a separator to the end of the compound string if *BooleanExpression* is True. If you omit the **SEPARATE** clause, the resulting string does not have a separator.

You cannot use imported or exported values as the operands of the **COMPOUND_STRING** function.

COMPOUND_STRING_TABLE(*StringExpression*[**,**...]**)**

A compound string table is an array of compound strings. Objects requiring a list of string values, such as the **XmNitems** and **XmNselectedItems** arguments for the **List** widget, use string table values. The **COMPOUND_STRING_TABLE** function builds the values for these two arguments of the **List** widget. The **COMPOUND_STRING_TABLE** function generates a value of type string_table. The name **STRING_TABLE** is a synonym for **COMPOUND_STRING_TABLE**.

The strings inside the string table can be simple strings, which the UIL compiler automatically converts to compound strings.

ASCIZ_STRING_TABLE(*StringExpression*[**,**...]**)**

An ASCIZ string table is an array of ASCIZ (null-terminated) string values separated by commas. This function allows you to pass more than one ASCIZ string as a callback tag value. The **ASCIZ_STRING_TABLE** function generates a value of type **asciz_table**. The name **ASCIZ_TABLE** is a synonym for **ASCIZ_STRING_TABLE**.

WIDE_CHARACTER(*StringExpression***)** Use the **WIDE_CHARACTER** function to generate a wide character string from a null-terminated string in the current locale.

CLASS_REC_NAME(*StringExpression***)** Use the **CLASS_REC_NAME** function to generate a widget class name. For a widget class defined by the toolkit, the string argument is the name of the class. For a user-defined widget, the string argument is the name of the creation function for the widget.

INTEGER_TABLE(*IntegerExpression*[**,**...]**)**

An integer table is an array of integer values separated by commas. This function allows you to pass more than one integer per callback tag value. The **INTEGER_TABLE** function generates a value of type **integer_table**.

ARGUMENTS(*StringExpression*[**,***ArgumentType*]**)**

The **ARGUMENTS** function defines the arguments to a user-defined widget. Each of the objects that can be described by UIL permits a set of arguments. For example, **XmNheight** is an argument to most objects and has the integer data type. To specify height for a user-defined widget, you can use the built-in argument name **XmNheight** and specify an integer value when you declare the user-defined widget. Do not use the **ARGUMENTS** function to specify arguments that are built into the UIL compiler.

The *StringExpression* name is the name the UIL compiler uses for the argument in the UID file. The *ArgumentType* is the type of value that can be associated with the argument. If you omit the second argument, the default type is **ANY** and no value type checking occurs. Use any of the following keywords to specify the argument type:

- **Any**
- **Asciz_Table**
- **Boolean**
- **Color**
- **Color_Table**
- **Compound_String**
- **Float**
- **Font**
- **Font_Table**
- **Fontset**
- **Icon**
- **Integer**
- **Integer_Table**
- **Reason**
- **Single_Float**
- **String**
- **String_Table**
- **Translation_Table**
- **Wide_Character**
- **WIdget**

You can use the **ARGUMENTS** function to allow the UIL compiler to recognize extensions to AIXwindows. For example, an existing widget can accept a new argument. Using the **ARGUMENTS** function, you can make this new argument available to the UIL compiler before the updated version of the compiler is released.

REASON(*StringExpression***)**

The **REASON** function is useful for defining new reasons for user-defined widgets.

Each of the objects in AIXwindows defines a set of conditions under which it calls a user-defined function. These conditions are known as *callback reasons*. The user-defined functions are called *callback procedures*. In a UIL module, you use a callbacks list to specify which user-defined functions are to be called for which reasons.

When you declare a user-defined widget, you can define callback reasons for that widget using the **REASON** function. The string expression specifies the argument name stored in the UID file for the reason. This reason name is supplied to the widget creation function at run time.

TRANSLATION_TABLE(*StringExpression*[**,**...]**)**

Each of the AIXwindows widgets have a translation table that maps X events (for example, pressing mouse button 1) to a sequence of actions. Through widget arguments, such as the common translations argument, you can specify an alternate set of events or actions for a particular widget. The **TRANSLATION_TABLE** function creates a translation table that can be used as the value of an argument that is of the data type translation_table.

You can use one of the following translation table directives with the **TRANSLATION_TABLE** function: **#override**, **#augment**, or **#replace**. The default is **#replace**. If you specify one of these directives, it must be the first entry in the translation table.

The **#override** directive causes any duplicate translations to be ignored. For example, if a translation for <**Btn1Down**> is already defined in the current translations for a PushButton, the translation defined by *NewTranslations* overrides the current definition. If the **#augment** directive is specified, the current definition takes precedence. The **#replace** directive replaces all current translations with those specified in the **XmNtranslations** resource.

Implementation Specifics

This file is part of the AIXwindows Development Environment in AIXwindows environment.

Files

/usr/include/uil/Uil.h

/usr/include/uil/UilDBDef.h

/usr/include/uil/UilDef.h

/usr/include/uil/UilSymDef.h

/usr/include/uil/UilSymGl.h

Related Information

The **uil** command.

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 attempts to logs in, the **login** program writes entries in two files:

- The **/etc/utmp** file, which contains a record of users logged into the system.
- The **/var/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.

The records in these files follow the **utmp** format, defined in the **utmp.h** header file.

Implementation Specifics

This file format is part of Accounting Services in the base operating system Extensions 2.

Files

/etc/utmp	Contains a record of users logged into the system.
/var/adm/wtmp	Contains connect accounting information.
/etc/security/failedlogin	Contains a record of invalid login attempts.

Related Information

The **fwtmp** command, **init** command, **login** command, **su** command, **who** command.

The **utmp.h** file, **lastlog** file format.

Accounting Commands in *AIX Version 4.3 System Management Concepts: Operating System and Devices* lists accounting commands that run automatically or keyboard commands entered from the keyboard.

Accounting Overview in *AIX Version 4.3 System Management Concepts: Operating System and Devices* and

vgrindefs File Format

Purpose

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

Description

The **vgrindefs** file format contains all the language definitions for the **vgrind** command. The database is very similar to the **terminfo** file format (file of terminal capabilities).

Fields

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

Name	Type	Description
`ab`	str	Alternate regular expression for the start of a comment.
`ae`	str	Alternate regular expression for the end of a comment.
`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	Presence means procedures are only defined at the top lexical level.
`oc`	bool	Presence means upper and lowercase are equivalent.
`kw`	str	List of keywords separated by spaces.

Examples

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:
```

The first field is the language name or any variants of the name. Thus the C language can be specified to the **vgrind** command in either lowercase or uppercase c.

Entries can continue onto multiple lines by giving a \ (backslash) as the last character of a line. The **vgrindefs** file format has the following two capabilities:

- Boolean capabilities that indicate a particular feature of the language
- String capabilities that give a regular expression or keyword list.

In Java, where comments can be delimited either by a starting "/*" or an ending "*", or by a starting "//" and "end" at the end of the line, the Java **vgrindefs** definition might be:

```
cb=/*:ce=*/:ab=//:ae=$
```

Regular Expressions

The **vgrindefs** file format uses regular expressions similar to those of the **ex** command and the **lex** command. The characters ^ (caret), $ (dollar sign), : (colon), and \ (backslash) are reserved characters and must be quoted with a preceding \ (backslash) if they are to be included as normal characters. The metasymbols and their meanings follow:

$ End of a line.

^ Beginning of a line.

\d Delimiter (space, tab, newline, start of line).

\a Matches any string of symbols, such as .* 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 (such as C) that can include the string delimiter in a string by escaping it.

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

Keyword List

The keyword list lists keywords in the language, separated by spaces. If the oc field is specified, indicating that uppercase and lowercase are equivalent, then all the keywords should be specified in lowercase.

Implementation Specifics

This file is part of Formatting Tools in the Text Formatting System.

Files

/usr/share/lib/vgrindefs	Contains terminal descriptions.

Related Information

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

WML File Format

Purpose

Generates variable UIL compiler components.

Description

The widget meta-language facility (WML) is used to generate changeable components of the user interface language (UIL) compiler, depending on the widget set. Using WML, you can add new widget UIL support to the AIXwindows widget set or add support for a totally new widget set.

File Format

WML files are ASCII files and can be modified with any standard text editor. They are accessed by WML in the **tools/wml** directory and have a **.wml** suffix. The Motif AIXwindows widget set is described in the **motif.wml** file. This is also the default WML file when using the WML facility.

When creating a WML file to add new widgets or change widget characteristics, you can make a copy of the **motif.wml** file and modify it. If you are creating a new widget set for use with UIL, create a completely new file. In either case, the **motif.wml** file is a good example of WML syntax and can help familiarize you with the language before attempting to write your own WML file.

WML files have a basic syntax that is similar in structure to UIL. WML syntax is made up of the following elements:

- Comments
- Data Type Definitions
- Character Set Definitions
- Enumeration Set Definitions
- Control List Definitions
- Class Definitions
- Child Definitions
- Resource Definitions

You can use spaces, tabs, or new-line characters anywhere in syntax, as long as they do not split keywords or strings. Comments end at a new-line character. The order of elements in syntax is not important.

The widget meta-language syntax examples shown use the following additional conventions:

[ ]	Indicates optional elements.
...	Indicates where an element of syntax can be repeated.
|	Indicates a choice among multiple items.

Comments

You can include comments in the WML file. Comments have the following syntax:

[*AnyElement*]**!***AnyComment*

Comments begin with an ! (exclamation point) and extend to the end of the line. A comment can begin on a line by itself or follow any part of another element. A comment does not change the meaning of any other element. For example:

```
!This is a comment
!   that spans two lines.
DataType   !This is a comment that follows code.
```

Data Type Definitions

Data type definitions register all the resource data types used in the file. You must register all the data types used in your WML file. Data type definitions have the following syntax:

DataType *AnyDatatype* [**{ InternalLiteral =** *InternalName* |
    DocName = "*String*"**;** [...]**}**]**;**
    [...]

A data type definition begins with the **DataType** keyword. Following the **DataType** keyword is a list of data types that can be modified with the following:

InternalLiteral Forces the value of the internal symbol table literal definition of the data type name. This modifier is used only to circumvent symbol table definitions hard-coded into the UIL compiler and should be used sparingly.

DocName Gives an arbitrary string for use in the documentation. This string supplies a different name for the data type or a single name for the data type if the data type has aliases.

For example:

```
DataType OddNumber {DocName="OddNumber";};
         NewString;
```

Character Set Definitions

Character set definitions register the AIXwindows Toolkit name and other information for the character set names used in UIL. Character set definitions have the following syntax:

CharacterSet
    AnyCharacterSet
        { [**FontListElementTag** | **XmStringCharsetName**] **=** "*String*"**;**
            [**Alias =** "*String*" ... **;** |
            Direction = [**LeftToRight** | **RightToLeft**] **;** |
            ParseDirection = [**LeftToRight** | **RightToLeft**] **;** |
            CharacterSize = [**OneByte** | **TwoByte**] **;**]
            [...] **} ;**
    [...]

A character set definition begins with the **CharacterSet** keyword. Following the **CharacterSet** keyword is a list of character sets that can be modified with the following:

FontListElementTag | XmStringCharsetName	Specifies the name of the character set. The set specified becomes the character set component of the compound string segment that is created. One of these character sets must be specified.
Alias	Specifies one or more aliases for the character set name. Each alias can be used within UIL to refer to the same character set.
Direction	Specifies the direction of a compound string segment created using this character set. The default is **LeftToRight**.
ParseDirection	Specifies the direction in which an input string is parsed when a compound string segment is created using this character set. If this is not specified, the value of **Direction** is the default.
CharacterSize	Specifies the number of bytes in each character of a compound string segment created using this character set. The default is **OneByte**.

An example of the character set definition syntax is as follows:

```
CharacterSet
   iso_latin1
                 { XmStringCharsetName = "ISO8859-1";
                                  Alias = "ISOLatin1"; } ;
   iso_hebrew_lr
                 { XmStringCharsetName = "ISO8859-8";
                                  Alias = "iso_latin8_lr";
                                  Direction = RightToLeft;
                                  ParseDirection = LeftToRight; } ;
   ksc_korean
                 { XmStringCharsetName = "KSC5601.1987-0";
                                  CharacterSize = TwoByte; };
```

Enumeration Set Definitions

Enumeration set definitions register the named constants used in the AIXwindows Toolkit to specify certain resource values. Enumeration set definitions have the following syntax:

EnumerationSet
ResourceName **:** *ResourceType*
{ *EnumerationValueName* **;** [...] **} ;**

An enumeration set definition begins with the **EnumerationSet** keyword. For each enumeration set defined, the name and type of the resource is listed. The resource name is the AIXwindows Toolkit resource name, with the beginning **XmN** prefix removed and the initial letter capitalized. For example, the name of the AIXwindows Toolkit resource **XmNrowColumnType** would be **RowColumnType**. The resource type is the data type for the resource; for most resources, this is the integer data type. Following the resource name and type is a list of enumeration value names that can be used as settings for the resource. These names are the same as those in the AIXwindows Toolkit.

An example of the enumeration set definition syntax is as follows:

```
EnumerationSet
   RowColumnType: integer
        { XmWORK_AREA; XmMENU_BAR; XmMENU_POPUP;
          XmMENU_PULLDOWN; XmMENU_OPTION; };
```

Control List Definitions

Control list definitions assign a name to groups of controls. You can use these control lists later in class definitions to simplify the structure of your WML file. Control list definitions have the following syntax:

ControlList
AnyControlList [**{** *AnyControl***;** [...]**}**]**;**

A control list definition starts with the **ControlList** keyword. Following the **ControlList** keyword are any number of control list definitions. Control list definitions are made up of a control list name followed by the set of controls it represents. For example:

```
ControlList
        Buttons {PushButton;
                RadioButton;
                CascadeButton;
                NewCascadebutton; } ;
```

Each control specified in the control list must be defined as a class in the file.

Class Definitions

Class definitions describe a particular widget class. Included in this description is its position in the class hierarchy, toolkit convenience function, resources, and controls. There should be one class definition for each widget or gadget in the widget set you want to support in UIL. Class definitions have the following syntax:

Class *ClassName* **: MetaClass** | **Widget** | **Gadget**
[**{**[
SuperClass = *ClassName***;** |
ParentClass = *ParentClassName***;** |
InternalLiteral = *InternalName***;** |
Alias = *Alias***;** |
ConvenienceFunction = *ConvenienceFunction***;** |
WidgetClass = *WidgetClass* **;** |
DocName = "*String***";** |
DialogClass = True | **False;** |

```
Resources { AnyResourceName [{
                    Default = NewDefaultValue; |
                    Exclude = True |
                    False;
                    [...]} ];
    [...]};|
Controls { AnyControlName; [...]};
Children { AnyChildName; [...]};
[...]
]}];
```

Class definitions start with the **Class** keyword. For each class defined, the name of the class and whether the class is a metaclass, widget, or gadget is listed. Each class definition can be modified using the following:

SuperClass	Indicates the name of the parent class. Only the root of the hierarchy does not specify a super class.
ParentClass	Indicates the name of the widget's automatically created parent class, if one exists. This allows resources for the automatically created parent class to be used in this class definition. For example, **XmBulletinBoardDialog** creates both an **XmBulletinBoard** and an **XmDialogShell**. To access the resources of the **XmDialogShell** parent class, specify it here.
InternalLiteral	Forces the value of the internal symbol table literal definition of the class name. This modifier is used only to circumvent symbol table definitions hard-coded into the UIL compiler and should be used sparingly.
Alias	Indicates alternate class names for use in a UIL specification.
ConvenienceFunction	Indicates the name of the creation convenience function for this class. All widget and gadget classes must have **ConvenienceFunction** specified.
WidgetClass	Indicates the associated widget class of gadget type classes. This value is currently not recognized.
DocName	Defines an arbitrary string for use in the documentation. This value is currently not recognized.
DialogClass	Indicates whether the class is a dialog class. This value is currently not recognized.
Resources	Lists the resources of the widget class. This keyword can be further modified with the following:
Default	Specifies a new default value for this resource. Resource default values are usually set in the resource definition. If an inherited resource's default value is changed by the class, the new default value should be noted here.
Exclude	Specifies whether an inherited resource should be excluded from the resource list of the class. The default value is False.
Children	Lists the names of the automatically created children of this class. This allows those children to be accessed in the UIL file.
Controls	Lists the controls that the widget class allows. The controls can be other classes or a control list from the control definition list.

An example of the usage of the preceding data type and control list definitions is shown:

```
Class
   TopLevelWidget : MetaClass
        {
        Resources
                {
                XtbNfirstResource;
                XtbNsecondResource;
                };
        };

   NewWidget : Widget
        {
        SuperClass = TopLevelWidget;
        ConvenienceFunction =
                XtbCreateNewWidget;
        Resources
        {
        XtbNnewResource;
        XtbNfirstResource
                {Default="XtbNEW_VALUE";};
        XtbNsecondResource
                {Exclude=True;};
        };
        Controls
        {
        NewWidget;
        Buttons;
        };
        };
```

Child Definitions

Child definitions register the classes of automatically created children. Automatically created children are referenced elsewhere in a UIL file using the **Children** keyword within a class definition. Child definitions have the following syntax:

Child
        ChildName **:** *ClassName***;**
        [...]

ChildName is the name of the automatically created child and *ClassName* is the name of the class of that child.

Resource Definitions

Resource definitions describe a particular resource. Included in this description is its type and default value. Each new resource reference in a class definition should have a resource definition. Resource definitions have the following syntax:

Resource
        ResourceName **: Argument** | **Reason** | **Constraint** | **SubResource**
                [**{**[
                Type = *Type* **;** |
                ResourceLiteral = *ResourceLiteral* **;** |
                InternalLiteral = *InternalName* **;** |
                Alias = *Alias* **;** |

Related = *Related* **;** |
Default = *Default* **;** |
DocName = *DocumentName* **;** |
[...]**}**]
[...]

Resource definitions start with the **Resource** keyword. For each resource definition, the name of the resource and whether the resource is an argument, reason, constraint, or subresource is listed.

Argument	Indicates a standard resource.
Reason	Indicates a callback resource.
Constraint	Indicates a constraint resource.
SubResource	This value is currently not recognized.

A resource definition can be modified with the following:

Type	Indicates the data type of the resource. The data type specified must be listed in the data type definition.
ResourceLiteral	Indicates the keyword used in the UIL file to reference the resource. In AIXwindows, the resource name is the same as the resource literal name (**ResourceLiteral**).
InternalLiteral	Forces the value of the internal symbol table literal definition of the resource name. This modifier is used only to circumvent symbol table definitions hard-coded into the UIL compiler and should be used sparingly.
Alias	Indicates alternate names for the resources used in a UIL specification.
Related	Special purpose field that allows resources that act as a counter for the current resources to be related to the resource. UIL automatically sets the value of this related resource to the number of items in the compiled instance of the *ResourceName* type.
Default	Indicates the default value of the resource.
DocName	Defines an arbitrary string for use in the documentation. This value is currently not recognized.

An example of the usage of data type, control list, and class definitions is shown:

```
Resource
   XtbNfirstResource : Argument
        { Type = OddNumber;
          Default = "XtbOLD_VALUE";};
   XtbNsecondResource : Argument
        { Type = NewString;
          Default = "XtbNEW_STRING";};
   XtbNnewResource : Argument
        { Type = OddNumber;
          Default = "XtbODD_NUMBER";};
```

Implementation Specifics

This file is part of the AIXwindows Development Environment in AIXwindows environment.

Related Information

XCOFF Object (a.out) File Format

Purpose

The extended common object file format (XCOFF) is the object file format for AIX. XCOFF combines the standard common object file format (COFF) with the TOC module format concept, which provides for dynamic linking and replacement of units within an object file. In AIX 4.3, XCOFF has been extended to provide for 64-bit object files and executable files.

XCOFF is the formal definition of machine-image object and executable files. These object files are produced by language processors (assemblers and compilers) and binders, and are used primarily by binders and the system loaders.

The default name for an XCOFF executable file is **a.out**.

Note: This information lists bits in big-endian order.

Read the following information to learn more about XCOFF object files:

- Composite File Header
- Sections and Section Headers
- Relocation Information for XCOFF File (reloc.h)
- Line Number Information for XCOFF File (linenum.h)
- Symbol Table Information
- dbx Stabstrings

Writing Applications that Use XCOFF Declarations

Programs can be written to understand 32-bit XCOFF files, 64-bit XCOFF files, or both. The programs themselves may be compiled in 32-bit mode or 64-bit mode to create 32-bit or 64-bit programs. By defining preprocessor macros, applications can select the proper structure definitions from the XCOFF header files.

Note: This document uses "XCOFF32" and "XCOFF64" as shorthand for "32-bit XCOFF" and "64-bit XCOFF", respectively.

Selecting XCOFF32 Declarations

To select the XCOFF32 definitions, an application merely needs to include the appropriate header files. Only XCOFF32 structures, fields, and preprocessor defines will be included. Structure names and field names will match those in previous versions of AIX, so existing programs can be recompiled on AIX 4.3 without change.

Note: Existing uses of shorthand type notation (e.g., `UINT, ULONG`) have been removed.

Selecting XCOFF64 Declarations

To select the XCOFF64 definitions, an application should define the preprocessor macro **__XCOFF64__**. When XCOFF header files are included, the structures, fields, and preprocessor defines for XCOFF64 will be included. Where possible, the structure names and field names are identical to the XCOFF32 names, but field sizes and offsets may differ.

Selecting Both XCOFF32 and XCOFF64 Declarations

To select structure definitions for both XCOFF32 and XCOFF64, an application should define both the preprocessor macros **__XCOFF32__** and **__XCOFF64__**. This will define structures for both kinds of XCOFF files. Structures and typedef names for XCOFF64 files will have the suffix "_64" added to them. (Consult the header files for details.)

Selecting Hybrid XCOFF Declarations

An application may choose to select single structures that contain field definitions for both XCOFF32 and XCOFF64 files. For fields that have the same size and offset in both XCOFF32 and XCOFF64 definitions, the field names are retained. For fields whose size or offset differ between XCOFF32 and XCOFF64 definitions, the XCOFF32 fields have a "32" suffix, while the XCOFF64 fields have a "64" suffix. To select hybrid structure definitions, an application should define the preprocessor macro **__XCOFF_HYBRID__**. For example, the symbol table definition (in **/usr/include/syms.h**) will have the names `n_offset32` and `n_offset64`, which should be used for the 32-bit XCOFF and 64-bit XCOFF respectively.

Understanding XCOFF

Assemblers and compilers produce XCOFF object files as output. The binder combines individual object files into an XCOFF executable file. The system loader reads an XCOFF executable file to create an executable memory image of a program. The symbolic debugger reads an XCOFF executable file to provide symbolic access to functions and variables of an executable memory image.

An XCOFF file contains the following parts:

- A composite header consisting of:
  - A file header
  - An optional auxiliary header
  - Section headers, one for each of the file’s raw-data sections
- Raw-data sections, at most one per section header
- Optional relocation information for individual raw-data sections
- Optional line number information for individual raw-data sections
- An optional symbol table
- An optional string table, which is used for all symbol names in XCOFF64 and for symbol names longer than 8 bytes in XCOFF32.

Not every XCOFF file contains every part. A minimal XCOFF file contains only the file header.

Object and Executable Files

XCOFF object files and executable files are similar in structure. An XCOFF executable file (or "module") must contain an auxiliary header, a loader section header, and a loader section.

The loader raw-data section contains information needed to dynamically load a module into memory for execution. Loading an XCOFF executable file into memory creates the following logical segments:

- A text segment (initialized from the `.text` section of the XCOFF file).
- A data segment, consisting of initialized data (initialized from the `.data` section of the XCOFF file) followed by uninitialized data (initialized by the system loader to 0). The length of uninitialized data is specified in the `.bss` section header of the XCOFF file.

The XCOFF file Organization illustrates the structure of the XCOFF object file.

XCOFF Header Files

The **xcoff.h** file defines the structure of the XCOFF file. The **xcoff.h** file includes the following files:

filehdr.h	Defines the file header.
aouthdr.h	Defines the auxiliary header.
scnhdr.h	Defines the section headers.
loader.h	Defines the format of raw data in the `.loader` section.
typchk.h	Defines the format of raw data in the `.typchk` section.
exceptab.h	Defines the format of raw data in the `.except` section.
dbug.h	Defines the format of raw data in the `.debug` section.
reloc.h	Defines the relocation information.
linenum.h	Defines the line number information.
syms.h	Defines the symbol table format.
storclass.h	Defines ordinary storage classes.
dbxstclass.h	Defines storage classes used by the symbolic debuggers.

The **a.out.h** file includes the **xcoff.h** file. All of the XCOFF include files include the **xcoff32_64.h** file.

For more information on sections of the XCOFF object file, see "Sections and Section Headers." For more information on the symbol table, see "Symbol Table Information." For more information on the string table, see "String Table." For more information on the Debug section, see "Debug Section."

Composite File Header

The following sections describe the XCOFF composite file header components:

- File Header (filehdr.h)
- Auxiliary Header (aouthdr.h)
- Section Headers (scnhdr.h)

File Header (filehdr.h)

The **filehdr.h** file defines the file header of an XCOFF file. The file header is 20 bytes long in XCOFF32 and 24 bytes long in XCOFF64. The structure contains the fields shown in the following table.

File Header Structure (Defined in filehdr.h)					
XCOFF32		**XCOFF64**			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	2	0	2	`f_magic`	Target machine
2	2	2	2	`f_nscns`	Number of sections
4	4	4	4	`f_timdat`	Time and date of file creation
8	4	8	8	`f_symptr`+	Byte offset to symbol table start
12	4	20	4	`f_nsyms`+	Number of entries in symbol table
16	2	16	2	`f_opthdr`	Number of bytes in optional header
18	2	18	2	`f_flags`	Flags (see "Field Definitions")
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.					

Field Definitions

`f_magic` Specifies an integer known as the *magic number*, which specifies the target machine and environment of the object file. For XCOFF32, the only valid value is `0x01DF (0737 Octal)`. For XCOFF64, the only valid value is `0x01EF (0757 Octal)`.

`f_nscns` Specifies the number of section headers contained in the file. The first section header is section header number one; all references to a section are one-based.

`f_timdat` Specifies when the file was created (number of elapsed seconds since 00:00:00 Universal Coordinated Time (UCT), January 1, 1970). This field should specify either the actual time or be set to a value of 0.

`f_symptr` Specifies a file pointer (byte offset from the beginning of the file) to the start of the symbol table. If the value of the `f_nsyms` field is 0, then this value is undefined.

`f_nsyms` Specifies the number of entries in the symbol table. Each symbol table entry is 18 bytes long.

`f_opthdr` Specifies the length, in bytes, of the auxiliary header. For an XCOFF file to be executable, the auxiliary header must exist and be **_AOUTHSZ_EXEC** bytes long. (**_AOUTHSZ_EXEC** is defined in **aouthdr.h**.)

`f_flags` Specifies a bit mask of flags that describe the type of the object file. The following information defines the flags:

Bit Mask	Flag
0x0001	F_RELFLG Indicates that the relocation information for binding has been removed from the file. This flag must not be set by compilers, even if relocation information was not required.
0x0002	F_EXEC Indicates that the file is executable. No unresolved external references exist.
0x0004	F_LNNO Indicates that line numbers have been stripped from the file by a utility program. This flag is not set by compilers, even if no line-number information has been generated.
0x0008	Reserved.
0x0010	Reserved.
0x0020	Reserved.
0x0040	Reserved.
0x0080	Reserved.
0x0100	Reserved.
0x0200	F_AR32W Indicates the byte ordering of a 32-bit-word big-endian architecture.
0x0400	F_PATCH Reserved.
0x1000	F_DYNLOAD Indicates the file is dynamically loadable and executable. External references are resolved by way of imports, and the file might contain exports and loader relocation.
0x2000	F_SHROBJ Indicates the file is a shared object (shared library). The file is separately loadable. That is, it is not normally bound with other objects, and its loader exports symbols are used as automatic import symbols for other object files.
0x4000	F_LOADONLY If the object file is a member of an archive, it can be loaded by the system loader, but the member is ignored by the binder. If the object file is not in an archive, this flag has no effect.

Auxiliary Header (aouthdr.h)

The auxiliary header contains system-dependent and implementation-dependent information, which is used for loading and executing a module. Information in the auxiliary header minimizes how much of the file must be processed by the system loader at execution time.

The binder generates an auxiliary header for use by the system loader. Auxiliary headers are not required for an object file that is not to be loaded. When auxiliary headers are generated by compilers and assemblers, the headers are ignored by the binder.

The auxiliary header immediately follows the file header.

Note: If the value of the `f_opthdr` field is 0, the auxiliary header does not exist.

The C language structure for the auxiliary header is defined in the **aouthdr.h** file. The auxiliary header contains the fields shown in the following table.

Auxiliary Header Structure (Defined in aouthdr.h)					
XCOFF32		XCOFF64		Name	Description
Offset	Length	Offset	Length		
0	2	0	2	`o_mflag`	Flags, how to execute
2	2	2	2	`o_vstamp`	Version
4	4	56	8	`o_tsize`+	Text size in bytes
8	4	64	8	`o_dsize`+	Initialized data size in bytes
12	4	72	8	`o_bsize`+	Uninitialized data size in bytes
16	4	80	8	`o_entry`+	Entry point descriptor (virtual address)
20	4	8	8	`o_text_start`+	Base address of text (virtual address)
24	4	16	8	`o_data_start`+	Base address of data (virtual address)
28	4	24	8	`o_toc`+	Address of TOC anchor
32	2	32	2	`o_snentry`	Section number for entry point
34	2	34	2	`o_sntext`	Section number for `.text`
36	2	36	2	`o_sndata`	Section number for `.data`
38	2	38	2	`o_sntoc`	Section number for TOC
40	2	40	2	`o_snloader`	Section number for loader data
42	2	42	2	`o_snbss`	Section number for `.bss`
44	2	44	2	`o_algntext`	Maximum alignment for `.text`
46	2	46	2	`o_algndata`	Maximum alignment for `.data`
48	2	48	2	`o_modtype`	Module type field
50	1	50	1	`o_cpuflag`	Bit flags - cpu types of objects
51	1	51	1	`o_cputype`	Reserved for CPU type
52	4	88	8	`o_maxstack`+	Maximum stack size allowed (bytes)
56	4	96	8	`o_maxdata`+	Maximum data size allowed (bytes)
60	4	4	4	`o_debugger`+	Reserved for debuggers.
64	8	52	4	`o_resv2`	Reserved Field must contain 0s.
N/A		104	116	`o_resv3`	Reserved. Field must contain 0s.

+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.

Field Definitions

The following information defines the auxiliary header fields. For entries with two labels, the label in parentheses is the alternate original COFF **a.out** file format name.

`o_mflags (magic)`	Specifies the magic number, which informs the operating system of the file's execution characteristics. The binder assigns the following value: **0x010B** Text and data are aligned in the file and may be paged.
`o_vstamp (vstamp)`	Specifies the format version for this auxiliary header. The only valid value is 1.
`o_tsize (tsize)`	Specifies the size (in bytes) of the raw data for the `.text` section. The `.text` section typically contains the read-only part of the program. This is the same value as contained in the `s_size` field of the section header for the `.text` section.
`o_dsize (dsize)`	Specifies the size (in bytes) of the raw data for the `.data` section. The `.data` section contains the initialized data of the program and is writable. This is the same value as contained in the `s_size` field of the section header for the `.data` section.
`o_bsize (bsize)`	Specifies the size (in bytes) of `.bss` area, which is used for uninitialized variables during execution and is writable. No raw data exists in the file for the `.bss` section. This is the same value as contained in the `s_size` field of the section header for the `.bss` section.
`o_entry (entry)`	Specifies the virtual address of the entry point. (See the definition of the `o_snentry` field.) For application programs, this virtual address is the address of the function descriptor. The function descriptor contains the addresses of both the entry point itself and its TOC anchor. The offset of the entry point function descriptor from the beginning of its containing section can be calculated as follows: `Section_offset_value=o_entry-s_paddr[o_snentry - 1],` where `s_paddr` is the virtual address contained in the section header.
`o_text_start (text_start)`	Specifies the virtual address of the .text section. This is the address assigned to (that is, used for) the first byte of the `.text` raw-data section. This is the same value as contained in the `s_paddr` field of the section header for the `.text` section.
`o_data_start (data_start)`	Specifies the virtual address of the `.data` section. This is the address assigned to (that is, used for) the first byte of the `.data` raw-data section. This is the same value as contained in the `s_paddr` field of the section header for the `.data` section. For addressing purposes, the `.bss` section is considered to follow the data section.

The following definitions are extensions used by the system loader. In general, an object file may contain multiple sections of a given type, but in a module, only a single occurrence of the `.text` , `.data` , `.bss` , and `.loader` sections may exist.

`o_toc`	Specifies the virtual address of the TOC anchor (see the definition of the `o_sntoc` field).
`o_snentry`	Specifies the number of the file section containing the entry-point. (This field contains a file section header sequence number.) The entry point must be in the `.text` or `.data` section.
`o_sntext`	Specifies the number of the file `.text` section. (This field contains a file section header sequence number.)
`o_sndata`	Specifies the number of the file `.data` section. (This field contains a file section header sequence number.)
`o_sntoc`	Specifies the number of the file section containing the TOC. (This field contains a file section header sequence number.)
`o_snloader`	Specifies the number of the file section containing the system loader information. (This field contains a file section header sequence number.)
`o_snbss`	Specifies the number of the file `.bss` section. (This field contains a file section header sequence number.)
`o_algntext`	Specifies the log (base 2) of the maximum alignment needed for any csect in the `.text` section.
`o_algndata`	Specifies the log (base 2) of the maximum alignment needed for any csect in the `.data` section and `.bss` sections.
`o_modtype`	Specifies a module type. The value is an ASCII character string. The following module type is recognized by the system loader: **RO** Specifies a read-only module. If a shared object with this module type has no BSS section and no dependents, the data section of the module will be mapped read-only and shared by all processes using the object.
`o_cpuflag`	Bit flags - cputypes of objects.
`o_cputype`	Reserved. This byte must be set to 0.
`o_maxstack`	Specifies the maximum stack size (in bytes) allowed for this executable. If the value is 0, the system default maximum stack size is used.
`o_maxdata`	Specifies the maximum data size (in bytes) allowed for this executable. If the value is 0, the system default maximum data size is used.
`o_debugger`	This field should contain 0. When a loaded program is being debugged, the memory image of this field may be modified by a debugger to insert a trap instruction.

Section Headers (scnhdr.h)

Each section of an XCOFF file has a corresponding section header, although some section headers may not have a corresponding raw-data section. A section header provides identification and file-accessing information for each section contained within an XCOFF file. Each section header in an XCOFF32 file is 40 bytes long, while XCOFF64 section headers are 72 bytes long. The C language structure for a section header can be found in the **scnhdr.h** file. A section header contains the fields shown in the following table.

Section Header Structure (Defined in scnhdr.h)					
XCOFF32		**XCOFF64**			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	8	0	8	`s_name`	Section name
8	4	8	8	`s_paddr`+	Physical address
12	4	16	8	`s_vaddr`+	Virtual address (same as physical address)
16	4	24	8	`s_size`+	Section size
20	4	32	8	`s_scnptr`+	Offset in file to raw data for section
24	4	40	8	`s_relptr`+	Offset in file to relocation entries for section
28	4	48	8	`s_lnnoptr`+	Offset in file to line number entries for section
32	2	56	4	`s_nreloc`+	Number of relocation entries
34	2	60	4	`s_nlnno`+	Number of line number entries
36	2	64	4	`s_flags`+	Flags to define the section type
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.					

Field Definitions

The following information defines the section header fields:

`s_name`
: Specifies an 8-byte, null-padded section name. An 8-byte section name will not have a terminating null character. Use the `s_flags` field instead of the `s_name` field to determine a section type. Two sections of the same type may have different names, allowing certain applications to distinguish between them.

`s_paddr`
: Specifies the physical address of the section. This is the address assigned and used by the compilers and the binder for the first byte of the section. This field should contain 0 for all sections except the `.text` , `.data` , and `.bss` sections.

`s_vaddr`
: Specifies the virtual address of the section. This field has the same value as the `s_paddr` field.

`s_size`
: Specifies the size (in bytes) of this section.

`s_scnptr`
: Specifies a file pointer (byte offset from the beginning of the file) to this section’s raw data. If this field contains 0, this section has no raw data. Otherwise, the size of the raw data must be contained in the `s_size` field.

`s_relptr`
: Specifies a file pointer (byte offset from the beginning of the file) to the relocation entries for this section. If this section has no relocation entries, this field must contain 0.

`s_lnnoptr` Specifies a file pointer (byte offset from the beginning of the file) to the line number entries for this section. If this section has no line number entries, this field must contain 0.

`s_nreloc` Specifies the number of relocation entries for this section. In an XCOFF32 file, if more than 65,534 relocation entries are required, the field value will be 65535, and an **STYP_OVRFLO** section header will contain the actual count of relocation entries in the `s_paddr` field. Refer to the discussion of overflow headers in "Sections and Section Headers" . If this field is set to 65535, the `s_nlnno` field must also be set to 65535.

`s_nlnno` Specifies the number of line number entries for this section. In an XCOFF32 file, if more than 65,534 line number entries are required, the field value will be 65535, and an **STYP_OVRFLO** section header will contain the actual number of line number entries in the `s_vaddr` field. Refer to the discussion of overflow headers in "Sections and Section Headers" . If this field is set to 65535, the `s_nreloc` field must also be set to 65535.

`s_flags` Specifies flags defining the section type. The low-order pair of bytes is used. A section type identifies the contents of a section and specifies how the section is to be processed by the binder or the system loader. Only a single bit value may be assigned to the `s_flags` field. This value must not be the sum or bitwise OR of multiple flags. The two high-order bytes should contain 0.

Valid bit values are:

Value	Flag
0x0000	Reserved.
0x0001	Reserved.
0x0002	Reserved.
0x0004	Reserved.
0x0008	STYP_PAD Specifies a pad section. A section of this type is used to provide alignment padding between sections within an XCOFF executable object file. This section header type is obsolete since padding is allowed in an XCOFF file without a corresponding pad section header.
0x0010	Reserved.
0x0020	STYP_TEXT Specifies an executable text (code) section. A section of this type contains the executable instructions of a program.
0x0040	STYP_DATA Specifies an initialized data section. A section of this type contains the initialized data and the TOC of a program.
0x0080	STYP_BSS Specifies an uninitialized data section. A section header of this type defines the uninitialized data of a program.
0x0100	STYP_EXCEPT Specifies an exception section. A section of this type provides information to identify the reason that a trap or exception occurred within an executable object program.
0x0200	STYP_INFO Specifies a comment section. A section of this type provides comments or data to special processing utility programs.
0x0400	Reserved.
0x0800	Reserved.
0x1000	STYP_LOADER Specifies a loader section. A section of this type contains object file information for the system loader to load an XCOFF executable. The information includes imported symbols, exported symbols, relocation data, type-check information, and shared object names.
0x2000	STYP_DEBUG Specifies a debug section. A section of this type contains stabstring information used by the symbolic debugger.
0x4000	STYP_TYPCHK Specifies a type-check section. A section of this type contains parameter/argument type-check strings used by the binder.
0x8000	STYP_OVRFLO **Note:** An XCOFF64 file may not contain an overflow section header. Specifies a relocation or line-number field overflow section. A section header of this type contains the count of relocation entries and line number entries for some other section. This section header is required when either of the counts exceeds 65,534. See the `s_nreloc` and `s_nlnno` fields in "Sections and Section Headers" for more information on overflow headers.

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

Sections and Section Headers

Section headers are defined to provide a variety of information about the contents of an XCOFF file. Programs that process XCOFF files will recognize only some of the valid sections.

See the following information to learn more about XCOFF file sections:

- Loader Section (loader.h)
- Debug Section
- Type-Check Section
- Exception Section
- Comment Section

Current applications do not use the `s_name` field to determine the section type. Nevertheless, conventional names are used by system tools, as shown in the following table.

Conventional Header Names

Description	Conventional Name	Multiple Allowed?	s_flag
Text section	`.text`	`Yes`	**STYP_TEXT**
Data section	`.data`	`Yes`	**STYP_DATA**
BSS section	`.bss`	`Yes`	**STYP_BSS**
Pad section	`.pad`	`Yes`	**STYP_PAD**
Loader section	`.loader`	`No`	**STYP_LOADER**
Debug section	`.debug`	`No`	**STYP_DEBUG**
Type-check section	`.typchk`	`Yes`	**STYP_TYPCHK**
Exception section	`.except`	`No`	**STYP_EXCEPT**
Overflow section	`.ovrflo`	`Yes` (one per `.text` or `.data` section)	**STYP_OVRFLO**
Comment section	`.info`	`Yes`	**STYP_INFO**

Some fields of a section header may not always be used, or may have special usage. This pertains to the following fields:

`s_name`
: On input, ignored by the binder and system loader. On output, the conventional names (shown in the "Conventional Header Names" table) are used.

`s_scnptr`
: Ignored for `.bss` sections.

`s_relptr` — Recognized for the `.text` and `.data` sections only. No relocation is performed for other sections, where this value must be 0.

`s_lnnoptr` — Recognized for the `.text` section only. Otherwise, it must be 0.

`s_nreloc` , `s_nlnno` — Handles relocation or line-number field overflows in an XCOFF32 file. (XCOFF64 files may not have overflow section headers.) If a section has more than 65,534 relocation entries or line number entries, both of these fields are set to a value of `65535`. In this case, an overflow section header with the `s_flags` field equal to **STYP_OVRFLO** is used to contain the relocation and line-number count information. The fields in the overflow section header are defined as follows:

- `s_nreloc` — Specifies the file section number of the section header that overflowed; that is, the section header containing a value of `65535` in its `s_nreloc` and `s_nlnno` fields. This value provides a reference to the primary section header. This field must have the same value as the `s_nlnno` field.

  Note: There is no reference in the primary section header that identifies the appropriate overflow section header. All the section headers must be searched to locate an overflow section header that contains the correct primary section header reference in this field.

- `s_nlnno` — Specifies the file section number of the section header that overflowed. This field must have the same value as the `s_nreloc` field.

- `s_paddr` — Specifies the number of relocation entries actually required. This field is used instead of the `s_nreloc` field of the section header that overflowed.

- `s_vaddr` — Specifies the number of line-number entries actually required. This field is used instead of the `s_nlnno` field of the section header that overflowed.

The `s_size` and `s_scnptr` fields have a value of 0 in an overflow section header. The `s_relptr` and `s_lnnoptr` fields must have the same values as in the corresponding primary section header.

An XCOFF file provides special meaning to the following sections:

- The `.text`, `.data`, and `.bss` sections define the memory image of the program. The relocation parts associated with the `.text` and `.data` sections contain the full binder relocation information so it can be used for replacement link editing. Only the `.text` section is associated with a line number part. The parts associated with the executable code are produced by the compilers and assemblers.
- The `.pad` section is defined as 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. Padding is allowed in an XCOFF file without a corresponding section header.
- The `.loader` section is a raw-data section defined to contain the dynamic loader information. This section is generated by the binder and has its own self-contained symbol table and relocation table. There is no reference to this section from the XCOFF Symbol Table.
- The `.debug` section is a raw-data section defined to contain the stab (symbol table) or dictionary information required by the symbolic debugger.
- The `.typchk` section is a raw-data section defined to contain parameter and argument type-checking strings.
- The `.except` section is a raw-data section defined to contain the exception tables used to identify the reasons for an exception in program execution.
- The `.info` comment section is a raw-data section defined to contain comments or data that are of significance to special processing utility programs.
- The `.debug`, `.except`, `.info`, and `.typchk` sections are produced by compilers and assemblers. References to these sections or to items within these sections are made from the XCOFF Symbol Table.

For more information on XCOFF file sections, see "Loader Section (loader.h)," "Debug Section," "Type-Check Section," "Exception Section," and "Comment Section."

Loader Section (loader.h)

The loader section contains information required by the system loader to load and relocate an executable XCOFF object. The loader section is generated by the binder. The loader section has an `s_flags` section type flag of **STYP_LOADER** in the XCOFF section header. By convention, `.loader` is the loader section name. The data in this section is not referenced by entries in the XCOFF symbol table.

The loader section consists of the following parts:

- Header fields
- Symbol table
- Relocation table
- Import file ID strings
- Symbol name string table

The C language structure for the loader section can be found in the **loader.h** file.

Loader Header Field Definitions

The following table describes the loader section’s header field definitions.

Loader Section Header Structure (Defined in loader.h)					
XCOFF32		XCOFF64		Name	Description
Offset	Length	Offset	Length		
0	4	0	4	`l_version`	Loader section version number
4	4	4	4	`l_nsyms`	Number of symbol table entries
8	4	8	4	`l_nreloc`	Number of relocation table entries
12	4	12	4	`l_istlen`	Length of import file ID string table
16	4	16	4	`l_nimpid`	Number of import file IDs
20	4	24	8	`l_impoff`+	Offset to start of import file IDs
24	4	20	4	`l_stlen`+	Length of string table
28	4	32	8	`l_stoff`+	Offset to start of string table
N/A		40	8	`l_symoff`	Offset to start of symbol table
N/A		48	8	`l_rldoff`	Offset to start of relocation entries
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.					

The following information defines the loader section's header fields:

`l_version` Specifies the loader section version number. This value must be 1 for XCOFF32, 2 for XCOFF64.

`l_nsyms` Specifies the number of symbol table entries in the loader section. This value is the actual count of symbol table entries contained in the loader section and does not include the three implicit entries for the `.text`, `.data`, and `.bss` symbol entries.

`l_nreloc` Specifies the number of relocation table entries in the loader section.

`l_istlen` Specifies the byte length of the import file ID string table in the loader section.

`l_nimpid` Specifies the number of import file IDs in the import file ID string table.

`l_impoff` Specifies the byte offset from beginning of the loader section to the first import file ID.

`l_stlen` Specifies the length of the loader section string table.

`l_stoff` Specifies the byte offset from beginning of the loader section to the first entry in the string table.

`l_symoff` Specifies the byte offset from beginning of the loader section to the start of the loader symbol table (in XCOFF64 only).

`l_rldoff` Specifies the byte offset from beginning of the loader section to the start of the loader section relocation entries (in XCOFF64 only).

Loader Symbol Table Field Definitions

The loader section symbol table contains the symbol table entries that the system loader needs for its import and export symbol processing and dynamic relocation processing.

The **loader.h** file defines the symbol table fields. Each entry is 24 bytes long.

There are three implicit external symbols, one each for the `.text`, `.data`, and `.bss` sections. These symbols are referenced using symbol table index values 0, 1, and 2, respectively. The first symbol contained in the loader section symbol table is referenced using an index value of 3.

Loader Section Symbol Table Entry Structure					
XCOFF32		**XCOFF64**		**Name**	**Description**
Offset	**Length**	**Offset**	**Length**		
0	8	N/A		`l_name`+	Symbol name or byte offset into string table
0	4	N/A		`l_zeroes`+	Zero indicates symbol name is referenced from `l_offset`
4	4	8	4	`l_offset`+	Byte offset into string table of symbol name
8	4	0	8	`l_value`+	Address field
12	2	12	2	`l_scnum`	Section number containing symbol
14	1	14	1	`l_smtype`	Symbol type, export, import flags
15	1	15	1	`l_smclas`	Symbol storage class
16	4	16	4	`l_ifile`	Import file ID; ordinal of import file IDs
20	4	20	4	`l_parm`	Parameter type-check field
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.					

The symbol table fields are:

`l_name` (XCOFF32 only) Specifies an 8-byte, null-padded symbol name if it is 8 bytes or less in length. Otherwise, the field is treated as the following two 4-byte integers for accessing the symbol name:

- `l_zeroes` (XCOFF32 only) A value of 0 indicates that the symbol name is in the loader section string table. This field overlays the first word of the `l_name` field. An `l_name` field having the first 4 bytes (first word) equal to 0 is used to indicate that the name string is contained in the string table instead of the `l_name` field.
- `l_offset` (XCOFF32 only) This field overlays the second word of the `l_name` field. The value of this field is the byte offset from the beginning of the loader section string table to the first byte of the symbol name (not its length field).

`l_offset` (XCOFF64 only) This field has the same use as the `l_offset` field in XCOFF32.

`l_value` Specifies the virtual address of the symbol

`l_scnum` Specifies the number of the XCOFF section that contains the symbol. If the symbol is undefined or imported, the section number is 0. Otherwise, the section number refers to the `.text`, `.data`, or `.bss` section. Section headers are numbered beginning with 1.

`l_smtype` Specifies the symbol type, import flag, export flag, and entry flag.

Bits 0-4 are flag bits defined as follows:

Bit 0	0x80	Reserved.
Bit 1	0x40	Specifies an imported symbol.
Bit 2	0x20	Specifies an entry point descriptor symbol.
Bit 3	0x10	Specifies an exported symbol.
Bit 4	0x08	Reserved.
Bits 5-7	0x07	Symbol type--see below.

Bits 5-7 constitute a 3-bit symbol type field with the following definitions:

0 XTY_ER

Specifies an external reference providing a symbol table entry for an external (global) symbol contained in another XCOFF object file.

1 XTY_SD

Specifies the csect section definition, providing the definition of the smallest initialized unit within an XCOFF object file.

2 XTY_LD

Specifies the label definition, providing the definition of the global entry points for initialized csects. An uninitialized csect of type **XTY_CM** may not contain a label definition.

3 XTY_CM

Specifies a common (BSS uninitialized data) csect definition, providing the definition of the smallest uninitialized unit within an XCOFF object file.

4-7 Reserved.

`l_smclas` Specifies the storage mapping class of the symbol, as defined in **syms.h** for the `x_smclas` field of the csect auxiliary symbol table entry. Values have the symbolic form XMC_*xx*, where *xx* is PR, RO, GL, XO, SV, SV64, SV3264, RW, TC, TD, DS, UA, BS, or UC. See "csect Auxiliary Entry for the **C_EXT** and **C_HIDEXT** Symbols" for more information.

l_ifile Specifies the import file ID string. This integer is the ordinal value of the position of the import file ID string in the import file ID name string table of the loader section. For an imported symbol, the value of 0 in this field identifies the symbol as a deferred import to the system loader. A deferred import is a symbol whose address can remain unresolved following the processing of the loader. If the symbol was not imported, this field must have a value of 0.

l_parm Specifies the offset to the parameter type-check string. The byte offset is from the beginning of the loader section string table. The byte offset points to the first byte of the parameter type-check string (not to its length field). For more information on the parameter type-check string, see "Type-Check Section" . A value of 0 in the l_parm field indicates that the parameter type-checking string is not present for this symbol, and the symbol will be treated as having a universal hash.

Loader Relocation Table Field Definitions

The Loader Section Relocation Table Structure contains all the relocation information that the system loader needs to properly relocate an executable XCOFF file when it is loaded. The **loader.h** file defines the relocation table fields. Each entry in the loader section relocation table is 12 bytes long in XCOFF32 and 16 bytes long in XCOFF64. The `l_vaddr`, `l_symndx`, and `l_rtype` fields have the same meaning as the corresponding fields of the regular relocation entries, which are defined in the **reloc.h** file. See "Relocation Information for XCOFF File (reloc.h)" for more information.

Loader Section Relocation Table Entry Structure					
XCOFF32		**XCOFF64**			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	4	0	8	`l_vaddr`+	Address field
4	4	12	4	`l_symndx`+	Loader section symbol table index of referenced item
8	2	8	2	`l_rtype`	Relocation type
10	2	10	2	`l_rsecnm`	File section number being relocated
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.					

The **loader.h** file defines the following fields:

Name	Description
`l_vaddr`	Specifies the virtual address of the relocatable reference.
`l_symndx`	Specifies the loader section symbol table index (*n*-th entry) of the symbol that is being referenced. Values 0, 1, and 2 are implicit references to the `.text`, `.data`, and `.bss` sections, respectively. Symbol index 3 is the index for the first symbol actually contained in the loader section symbol table. **Note:** A reference to an exported symbol can be made using the symbol's section number (symbol number 0, 1, or 2) or using the actual number of the exported symbol.
`l_rtype`	Specifies the relocation size and type. (This field has the same interpretation as the `r_type` field in the **reloc.h** file.) See "Relocation Information for XCOFF File (reloc.h)" for more information.
`l_rsecnm`	Specifies the section number of the `.text`, `.data`, or `.bss` section being relocated (associated with `l_vaddr` field). This is a one-based index into the section headers.

Loader Import File ID Name Table Definition

The loader section import file ID name strings of a module provide a list of dependent modules that the system loader must load in order for the module to load successfully. However, this list does not contain the names of modules that the named modules themselves depend on.

Loader Section Import File IDs - Contains Variable Length Strings			
Offset	**Length in Bytes**	**Name**	**Description**
0	*n*1	`l_impidpath`	Import file ID path string, null-delimited
*n*1 + 1	*n*2	`l_impidbase`	Import file ID base string, null-delimited
*n*1 + *n*2 + 2	*n*3	`l_impidmem`	Import file ID member string, null-delimited
			Fields repeat for each import file ID.

Each import file ID name consists of three null-delimited strings.

The first import file ID is a default **LIBPATH** value to be used by the system loader. 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 bytes.

Each entry in the import file ID name table consists of:

- Import file ID path name
- Null delimiter (ASCII Null Character)
- Import file ID base name
- Null delimiter
- Import file ID archive-file-member name
- Null delimiter

For example:

```
/usr/lib\0mylib.a\0shr.o\0
```

Loader String Table Definition

The loader section string table contains the parameter type-checking strings, all symbols names for an XCOFF64 file, and the names of symbols longer than 8 bytes for an XCOFF32 file. Each string consists of a 2-byte length field followed by the string.

Loader Section String Table		
Offset	**Length in Bytes**	**Description**
0	2	Length of string.
2	*n*	Symbol name string (null-delimited) or parameter type string (not null-delimited).
		Fields repeat for each string.

Symbol names are null-terminated. The value in the length-field includes the length of the string plus the length of the null terminator but does not include the length of the length field itself.

The parameter type-checking strings contain binary values and are not null-terminated. The value in the length field includes the length of the string only but does not include the length of the length field itself.

The symbol table entries of the loader section contain a byte offset value that points to the first byte of the string instead of to the length field.

Loader Section Header Contents

The contents of the section header fields for the loader section are:

Name	Contents
`s_name`	**.loader**
`s_paddr`	0
`s_vaddr`	0
`s_size`	The size (in bytes) of the loader section
`s_scnptr`	Offset from the beginning of the XCOFF file to the first byte of the loader section data
`s_relptr`	0
`s_lnnoptr`	0
`s_nreloc`	0
`s_nlnno`	0
`s_flags`	**STYP_LOADER**

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

For more information on XCOFF file sections, see "Sections and Section Headers," "Debug Section," "Type-Check Section," "Exception Section," and "Comment Section."

Debug Section

The debug section contains the symbolic debugger stabstrings (symbol table strings). It is generated by the compilers and assemblers. It provides symbol attribute information for use by the symbolic debugger. The debug section has a section type flag of **STYP_DEBUG** in the XCOFF section header. By convention, `.debug` is the debug section name. The data in this section is referenced from entries in the XCOFF symbol table. A stabstring is a null-terminated character string. Each string is preceded by a 2-byte length field in XCOFF32 or a 4-byte length field in XCOFF64.

Field Definitions

The following two fields are repeated for each symbolic debugger stabstring:

- A 2-byte (XCOFF32) or 4-byte (XCOFF64) length field containing the length of the string. The value contained in the length field includes the length of the terminating null character but does not include the length of the length field itself.
- The symbolic debugger stabstring.

Refer to discussion of symbolic debugger stabstring grammar for the specific format of the stabstrings.

Debug Section Header Contents

The contents of the section header fields for the debug section are:

Name	Contents
`s_name`	**.debug**
`s_paddr`	0
`s_vaddr`	0
`s_size`	The size (in bytes) of the debug section
`s_scnptr`	Offset from the beginning of the XCOFF file to the first byte of the debug section data
`s_relptr`	0
`s_lnnoptr`	0
`s_nreloc`	0
`s_nlnno`	0
`s_flags`	**STYP_DEBUG**

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

For more information on XCOFF file sections, see "Sections and Section Headers," "Debug Section," "Type-Check Section,", "Exception Section," and "Comment Section."

Type-Check Section

The type-check section contains the type-checking hash strings and is produced by compilers and assemblers. It is used by the binder to detect variable mismatches and argument interface errors when linking separately compiled object files. (The type-checking hash strings in the loader section are used to detect these errors prior to running a program.) The type-check section has a section type flag of **STYP_TYPCHK** in the XCOFF section header. By convention, `.typchk` is the type-check section name. The strings in this section are referenced from entries in the XCOFF symbol table.

Field Definitions

The following two fields are repeated for each parameter type-checking string:

- A 2-byte length field containing the length of the type-checking string. The value contained in the length field does not include the length of the length field itself.
- The parameter type-checking hash string.

Type Encoding and Checking Format for Data

The type-checking hash strings are used to detect errors prior to execution of a program. Information about all external symbols (data and functions) is encoded by the compilers and then checked for consistency at bind time and load time. The type-checking strings are designed to enforce the maximum checking required by the semantics of each particular language supported, as well as 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 uses a unique value, UNIVERSAL, that matches any code. The UNIVERSAL hash can be used as an escape mechanism for assembly programs or for programs in which type information or subroutine interfaces might not be known. The UNIVERSAL hash is four blank ASCII characters (0x20202020) or four null characters (0x00000000).

The following 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 code representing each language. These codes are the same as those defined for the `e_lang` field in the "Exception Section" information .

general hash
: A 4-byte field representing the most general form by which a data symbol or function can be described. This form is the most common to languages supported by . If the information is incomplete or unavailable, a universal hash should be generated. The general hash is language-independent and must match 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 required by a given language. This part is used in intra-language binding and is not checked unless both symbols have the same language identifier.

Section Header Contents

The contents of the section header fields for the type-check section are:

Name	Contents
`s_name`	**.typchk**
`s_paddr`	0
`s_vaddr`	0
`s_size`	The size (in bytes) of the type-check section
`s_scnptr`	Offset from the beginning of the XCOFF file to the first byte of the type-check section data
`s_relptr`	0
`s_lnnoptr`	0
`s_nreloc`	0
`s_nlnno`	0
`s_flags`	**STYP_TYPCHK**.

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

For more information on XCOFF file sections, see "Sections and Section Headers," "Debug Section," "Type-Check Section," "Exception Section," and "Comment Section."

Exception Section

The exception section contains addresses of trap instructions, source language identification codes, and trap reason codes. This section is produced by compilers and assemblers, and used during or after run time to identify the reason that a specific trap or exception occurred. The exception section has a section type flag of **STYP_EXCEPT** in the XCOFF section header. By convention, `.except` is the exception section name. Data in the exception section is referenced from entries in the XCOFF symbol table.

An exception table entry with a value of 0 in the `e_reason` field contains the symbol table index to a function's **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. For more information on this entry, see "csect Auxiliary Entry for **C_EXT** and **C_HIDEXT** Symbols" .

The C language structure for the exception section entries can be found in the **exceptab.h** file.

The exception section entries contain the fields shown in the following tables.

Initial Entry: Exception Section Structure					
XCOFF32		**XCOFF64**			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	4	0	4	`e_addr.e_symndx`+	Symbol table index for function
4	1	8	1	`e_lang`+	Compiler language ID code
5	1	9	1	`e_reason`+	Value 0 (exception reason code 0)
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined. With e_addr.e_symndx, the suffix is added to e_addr (i.e. e_addr32.e_symndx).					

Subsequent Entry: Exception Section Structure					
XCOFF32		XCOFF64			
Offset	Length	Offset	Length	Name	Description
0	4	0	8	`e_addr.e_paddr`+	Address of the trap instruction
4	1	8	1	`e_lang`+	Compiler language ID code
5	1	9	1	`e_reason`+	Trap exception reason code
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined. With e_addr.e_paddr, the suffix is added to e_addr (i.e. e_addr32.e_paddr).					

Field Definitions

The following defines the fields listed of the exception section:

`e_symndx`	Contains an integer (overlays the `e_paddr` field). When the `e_reason` field is 0, this field is the symbol table index of the function.
`e_paddr`	Contains a virtual address (overlays the `e_symndx` field). When the `e_reason` field is nonzero, this field is the virtual address of the trap instruction.
`e_lang`	Specifies the source language. The following list defines the possible values of the `e_lang` field.

ID	Language
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
0x0D-0xFF	Reserved

`e_reason`	Specifies an 8-bit, compiler-dependent trap exception reason code. Zero is not a valid trap exception reason code because it indicates the start of exception table entries for a new function.

Section Header Contents

The following fields are the contents of the section header fields for the exception section.

Name	Contents
s_name	**.except**
s_paddr	0
s_vaddr	0
s_size	The size (in bytes) of the exception section
s_scnptr	Offset from the beginning of the XCOFF file to the first byte of the exception section data
s_relptr	0
s_lnnoptr	0
s_nreloc	0
s_nlnno	0
s_flags	**STYP_EXCEPT**

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

For more information on XCOFF file sections, see "Sections and Section Headers," "Debug Section," "Type-Check Section," "Exception Section," and "Comment Section."

Comment Section

The comment section contains information of special processing significance to an application. This section can be produced by compilers and assemblers and used during or after run time to fulfill a special processing need of an application. The comment section has a section type flag of **STYP_INFO** in the XCOFF section header. By convention, `.info` is the comment section name. Data in the comment section is referenced from **C_INFO** entries in the XCOFF symbol table.

The contents of a comment section consists of repeated instances of a 4-byte `length` field followed by a string of bytes (containing any binary value). The length of each string is stored in its preceding 4-byte `length` field. The string of bytes need not be terminated by a null character nor by any other special character. The specified length does not include the length of the `length` field itself. A length of 0 is allowed. The format of the string of bytes is not specified.

A comment section string is referenced from an entry in the XCOFF symbol table. The storage class of the symbol making a reference is **C_INFO**. See "Symbol Table Field Contents by Storage Class" for more information.

A **C_INFO** symbol is associated with the nearest **C_FILE**, **C_EXT**, or **C_HIDEXT** symbol preceding it.

Section Header Contents

The following fields are the contents of the section header fields for the comment section.

Name	Contents
`s_name`	**.info**
`s_paddr`	0
`s_vaddr`	0
`s_size`	The size (in bytes) of the comment section
`s_Scnptr`	Offset from the beginning of the XCOFF file to the first byte of the comment section data
`s_relptr`	0
`s_lnnoptr`	0
`s_nreloc`	0
`s_nlnno`	0
`s_flags`	**STYP_INFO**

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

For more information on XCOFF file sections, see "Sections and Section Headers," "Debug Section," "Type-Check Section," "Exception Section," and "Comment Section."

Relocation Information for XCOFF File (reloc.h)

The `.text` section and `.data` section may have relocation information. The relocation information is used by the binder to modify the `.text` section and `.data` section contents with address and byte-offset information of individual XCOFF object files collected into an XCOFF executable file.

The compilers and assemblers are responsible for generating the relocation entries for the `.text` and `.data` sections.

The binder generates relocation information for the `.loader` section, as required by the system loader.

Each relocation entry of the `.text` and `.data` section is 10 bytes long (14 for XCOFF64). (A relocation entry in the `.loader` section is 12 bytes long (16 for XCOFF64) and is explained in the loader section description in this document. See "Relocation Table Field Definitions" for more information.) The C language structure for a relocation entry can be found in the **reloc.h** file. A relocation entry contains the fields shown in the following table.

Relocation Entry Structure					
XCOFF32		**XCOFF64**		**Name**	**Description**
Offset	**Length**	**Offset**	**Length**		
0	4	0	8	`r_vaddr`+	Virtual address (position) in section to be relocated
4	4	8	4	`r_symndx`+	Symbol table index of item that is referenced
8	1	12	1	`r_rsize`+	Relocation size and information
9	1	13	1	`r_rtype`+	Relocation type
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.					

The relocation entries for the `.text` and `.data` sections are part of their respective sections. The relocation entry refers to a location to be modified. The relocation entries for a section must be in ascending address order.

(The loader section contains a single set of relocation entries used by the system loader, so a section number is required within each relocation entry to identify the section that needs to be modified.)

Field Definitions

The following defines the relocation-information fields:

`r_vaddr`
: Specifies the virtual address of the value that requires modification by the binder. The byte offset value to the data that requires modification from the beginning of the section that contains the data can be calculated as follows:

```
offset_in_section = r_vaddr - s_paddr
```

`r_symndx`
: Specifies a zero-based index into the XCOFF symbol table for locating the referenced symbol. The symbol table entry contains an address used to calculate a modification value to be applied at the `r_vaddr` relocation address.

`r_rsize`
: Specifies the relocation size and sign. Its contents are detailed in the following list:

0x80 (1 bit)	Indicates whether the relocation reference is signed (1) or unsigned (0).
0x40 (1 bit)	If this field is one, it indicates that the binder replaced the original instruction by a branch instruction to a special fixup instruction sequence.
0x3F(6 bits)	Specifies the bit length of the relocatable reference minus one. The current architecture allows for fields of up to 32 bits (XCOFF32) or 64 bits (XCOFF64) to be relocated.

`r_rtype` Specifies an 8-bit relocation type field that indicates to the binder which relocation algorithm to use for calculating the modification value. This value is applied at the relocatable reference location specified by the `r_vaddr` field. The following relocation types are defined:

0x00 R_POS

Specifies positive relocation. Provides the address of the symbol specified by the `r_symndx` field.

0x01 R_NEG

Specifies negative relocation. Provides the negative of the address of the symbol specified by the `r_symndx` field.

0x02 R_REL

Specifies relative-to-self relocation. Provides a displacement value between the address of the symbol specified by the `r_symndx` field and the address of the csect to be modified.

0x03 R_TOC

Specifies relative-to-TOC relocation. Provides a displacement value that is the difference between the address value in the symbol specified by the `r_symndx` field and the address of the TOC anchor csect. The TOC anchor csect has a symbol table csect auxiliary entry with an `x_smclass` (storage mapping class) value of **XMC_TC0**. The TOC anchor csect must be of zero length. There may be only one TOC anchor csect per XCOFF section.

0x04 R_TRL

Specifies TOC Relative Indirect Load (modifiable) relocation. Provides a displacement value that is the difference between the address value in the symbol specified by the `r_symndx` field and the address of the TOC anchor csect. This relocation entry is treated the same as an **R_TOC** relocation entry. It provides the following additional information concerning the instruction being relocated: The instruction that is referenced by the `r_vaddr` field is a load instruction. That load instruction is permitted to be modified by the binder to become a compute address instruction. Changing an instruction from a load instruction to a compute address instruction avoids a storage reference during execution. A compute address instruction can be used if the address contained at the address specified by the `r_symndx` field has a value that itself references a `r_symndx` field that can be accessed with a valid in-range displacement relative to the TOC anchor address. That is, the target of the TOC entry is from -32,768 to 32,767, inclusive, from the TOC anchor address. If a compute address instruction is generated by the binder, the **R_TRL** relocation type is changed to become a **R_TRLA** type. This allows the reverse transformation, if required. Compilers are permitted to generate this relocation type.

0x13 R_TRLA

Specifies TOC Relative Load Address (modifiable LA to L) relocation. Provides a displacement value that is the difference between the address value in the symbol specified by the `r_symndx` field and the address of the TOC anchor csect. This relocation entry is treated the same as an **R_TOC** relocation entry. It provides the following additional information concerning the instruction being relocated: The instruction that is referenced by the `r_vaddr` field is a compute address instruction. The compute address instruction is modified by the binder to become a load instruction whenever the calculated displacement value is outside the valid displacement range relative to the TOC anchor address. This relocation type provides the binder with a means to transform a compute address instruction into a load instruction whenever required. If a load instruction is generated by the binder, the **R_TRLA** relocation type is changed to become an **R_TRL** type. Compilers are not permitted to generate this relocation type.

0x05 R_GL

Specifies Global Linkage-External TOC address relocation. Provides the address of the TOC associated with a defined external symbol. The external symbol with the required TOC address is specified by the `r_symndx` field of the relocation entry. This relocation entry provides a method of accessing the address of the TOC contained within the same executable where the `r_symndx` external symbol is defined.

0x06 R_TCL

Specifies local object TOC address relocation. Provides the address of the TOC associated with a defined external symbol. The external symbol for which the TOC address is required is specified by the `r_symndx` field of the relocation entry. The external symbol is defined locally within the resultant executable. This relocation entry provides a method of accessing the address of the TOC contained within the same executable where the `r_symndx` external symbol is defined.

0x0C R_RL

Treated the same as the **R_POS** relocation type.

0x0D R_RLA

Treated the same as the **R_POS** relocation type.

0x0F R_REF

Specifies a nonrelocating reference to prevent garbage collection (by the binder) of a symbol. This relocation type is intended to provide compilers and assemblers a method to specify that a given csect has a dependency upon another csect without using any space in the actual csect. The reason for making the dependency reference is to prevent the binder from garbage-collecting (eliminating) a csect for which another csect has an implicit dependency.

0x08 R_BA

Treated the same as the **R_RBA** relocation type.

0x18 R_RBA

Specifies branch absolute relocation. Provides the address of the symbol specified by the `r_symndx` field as the target address of a branch instruction. The instruction can be modified to a (relative) branch instruction if the target address is relocatable.

0x0A R_BR

Treated the same as the **R_RBR** relocation type.

0x1A R_RBR

Specifies (relative) branch relocation. Provides a displacement value between the address of the symbol specified by the `r_symndx` field and the address of the csect containing the branch instruction to be modified. The instruction can be modified to an absolute branch instruction if the target address is not relocatable.

The **R_RBR** relocation type is the standard branch relocation type used by compilers and assemblers for the . This relocation type along with **glink** code allows an executable object file to have a text section that is position-independent.

Additional Relocation Features

Standard practice is to retain relocation information only for unresolved references or references between distinct sections. Once a reference is resolved, the relocation information is discarded. This is sufficient for an incremental bind and a fixed address space model. To provide the capability for rebinding and handling a relocatable address space model, the relocation information is not discarded from an XCOFF file.

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

For more information on relocation field table definitions, see "Relocation Table Field Definitions" in the loader section.

Line Number Information for XCOFF File (linenum.h)

Line number entries are used by the symbolic debugger to debug code at the source level. When present, there is a single line number entry for every source line that can have a symbolic debugger breakpoint. The line numbers are grouped by function. The beginning of each function is identified by the `l_lnno` field containing a value of 0. The first field, `l_symndx` , is the symbol table index to the **C_EXT** or **C_HIDEXT** symbol table entry for the function.

Each line number entry is six bytes long. The C language structure for a line number entry can be found in the **linenum.h** file. A line number entry contains the fields shown in the following tables.

Initial Line Number Structure Entry for Function					
XCOFF32		**XCOFF64**			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	4	0	4	`l_addr.l_symndx`+	Symbol table index for function
4	2	8	4	`l_lnno`+	Value 0 (line number 0)
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined. With l_addr.l_symndx, the suffix is added to l_addr (i.e. l_addr32.l_symndx).					

Subsequent Line Number Entries for Function					
XCOFF32		**XCOFF64**			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	4	0	8	`l_paddr`+	Address at which break point can be inserted
4	2	8	4	`l_lnno`+	Line number relative to start of function
+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined. With l_addr.l_paddr, the suffix is added to l_addr (i.e. l_addr32.l_paddr).					

Field Definitions

The following list defines the line number entries:

`l_symndx`	Specifies the symbol table index to the function name (overlays the `l_paddr` field). When the `l_lnno` field is 0, this interpretation of the field is used.
`l_paddr`	Specifies the virtual address of the first instruction of the code associated with the line number (overlays the `l_symndx` field). When the `l_lnno` field is not 0, this interpretation of the field is used.
`l_lnno`	Specifies either the line number relative to the start of a function or 0 to indicate the beginning of a function.

Note: If part of a function other than the beginning comes from an include file, the line numbers are absolute, rather than relative to the beginning of the function. (See the **C_BINCL** and **C_EINCL** symbol types in "Storage Classes by Usage and Symbol Value Classification" for more information.)

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

For information on debugging, see "Debug Section."

Symbol Table Information

One composite symbol table is defined for an XCOFF file. The symbol table contains information required by both the binder (external symbols) and the symbolic debugger (function definitions and internal and external symbols).

The symbol table consists of a list of 18-byte, fixed-length entries. Each symbol represented in the symbol table consists of at least one fixed-length entry, and some are followed by auxiliary entries of the same size.

See the following information to learn more about the symbol table:

- Symbol Table Auxiliary Information
- Symbol Table Field Contents by Storage Class
- String Table

For each external symbol, one or more auxiliary entries are required that provide additional information concerning the external symbol. There are three major types of external symbols of interest to the binder, performing the following functions:

- Define replaceable units or csects.
- Define the external names for functions or entry points within csects.
- Reference the names of external functions in another XCOFF object.

For symbols defining a replaceable unit (csect), a csect auxiliary entry defines the length and storage-mapping class of the csect. For symbols defining external names for functions within a csect, the csect auxiliary entry points to the containing csect, the parameter type-checking information, and the symbolic debugger information for the function. For symbols referencing the name of an external function, a csect auxiliary entry identifies the symbol as an external reference and points to parameter type-checking information.

Symbol Table Contents

An XCOFF symbol table has the following general contents and ordering:

- The **C_FILE** symbol table entries used to bracket all the symbol table entries associated with a given source file.
- The **C_INFO** comment section symbol table entries that are of source file scope. These follow the **C_FILE** entry but before the first csect definition symbol table entry.
- The symbolic debugger symbol table entries that are of file scope. These follow the **C_FILE** entry but before the first csect entry.
- csect definition symbol table entries used to define and bracket all the symbols contained with a csect.
- **C_INFO** comment section symbol table entries that follow a csect definition symbol table entry are associated with that csect.
- All symbolic debugger symbol table entries that follow a csect definition symbol table entry or label symbol table entry are associated with that csect or label.

The ordering of the symbol table must be arranged by the compilers and assemblers both to accommodate the symbolic debugger requirements and to permit effective management by the binder of the different sections of the object file as a result of such binder actions as garbage collection, incremental binding, and rebinding. This ordering is required by the binder so that if a csect is deleted or replaced, all the symbol table information associated with the csect can also be deleted or replaced. Likewise, if all the csects associated with a source file are deleted or replaced, all the symbol table and related information associated with the file can also be deleted or replaced.

Symbol Table Layout

The following example shows the general ordering of the symbol table.

```
un_external      Undefined global symbols

.file                Prolog --defines stabstring compaction level
.file                Source file 1
  .info              Comment section reference symbol with file scope
  stab               Global Debug symbols of a file
  csect              Replaceable unit definition (code)
     .info           Comment section reference symbol with csect scope
     function        Local/External function
         stab        Debug and local symbols of function
     function        Local/External function
         stab        Debug and local symbols of function
  ..............
  csect              Replaceable unit definition (local statics)
         stab        Debug and local statics of file
  ..............
  csect              Relocatable unit definition (global data)
         external    Defined global symbol
         stab        Debug info for global symbol
  ..............
.file                Source file 2
  stab               Global Debug symbols of a file
```

```
  csect              Replaceable unit definition (code)
         function    Local/External function
             stab    Debug and local symbols of function
  ..............
  csect              Replaceable unit definition (local statics)
      stab           Debug and Local statics of file
  ..............
  csect              Replaceable unit definition (global data)
      external       Defined global symbol
          stab       Debug info for global symbol
.file                Source file
  ..............
```

Symbol Table Entry (syms.h)

Each symbol, regardless of storage class and type, has a fixed-format entry in the symbol table. In addition, some symbol types may have additional (auxiliary) symbol table entries immediately following the fixed-format entry. Each entry in the symbol table is 18 bytes long. The C language structure for a symbol table entry can be found in the **syms.h** file. The index for the first entry in the symbol table is 0. The following table shows the structure of the fixed-format part of each symbol in the symbol table.

Symbol Table Entry Format

XCOFF32		XCOFF64			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	8	N/A		`n_name`	Symbol name (occupies the same 8 bytes as `n_zeroes` and `n_offset`)
0	4	N/A		`n_zeroes`	Zero, indicating name in string table or `.debug` section (overlays first 4 bytes of `n_name`)
4	4	8	4	`n_offset`+	Offset of the name in string table or `.debug` section (In XCOFF32: overlays last 4 bytes of `n_name`)
8	4	0	8	`n_value`+	Symbol value; storage class-dependent
12	2	12	2	`n_scnum`	Section number of symbol
14	2	14	2	`n_type`	Basic and derived type specification
14	1	14	1	`n_lang`	Source language ID (overlays first byte of n_type)
15	1	15	1	`n_cpu`	CPU Type ID (overlays second byte of n_type)
16	1	16	1	`n_sclass`	Storage class of symbol
17	1	17	1	`n_numaux`	Number of auxiliary entries

+Use "32" or "64" suffix when **__XCOFF_HYBRID__** is defined.

Field Definitions

The following defines the symbol table entry fields:

`n_name`
: Used by XCOFF32 only. Specifies an 8-byte, null-padded symbol name or symbolic debugger stabstring. The storage class field is used to determine if the field is a symbol name or symbolic debugger stabstring. By convention, a storage class value with the high-order bit on indicates that this field is a symbolic debugger stabstring.

  If the XCOFF32 symbol name is longer than 8 bytes, the field is interpreted as the following two fields:

  `n_zeroes`
  : A value of 0 indicates that the symbol name is in the string table or `.debug` section (overlays first word of `n_name`).

  `n_offset`
  : Specifies the byte offset to the symbol name in the string table or `.debug` section (overlays last 4 bytes of `n_name`). The byte offset is relative to the start of the string table or `.debug` section. A byte offset value of 0 is a null or zero-length symbol name.

`n_offset`
: For XCOFF64: Specifies the byte offset to the symbol name in the string table or `.debug` section. The byte offset is relative to the start of the string table or `.debug` section. A byte offset value of 0 is a null or zero-length symbol name. (For XCOFF32 only, used in conjunction with `n_zeroes`. See entry immediately above.)

`n_value`
: Specifies the symbol value. The contents of the symbol value field is storage class-dependent, as shown in the following definitions:

Content	**Storage Class**
Relocatable address	**C_EXT, C_HIDEXT, C_FCN, C_BLOCK, C_STAT**
Zero	**C_GSYM, C_BCOMM, C_DECL, C_ENTRY, C_ESTAT, C_ECOMM**
Offset in csect	**C_FUN, C_STSYM**
Offset in file	**C_BINCL, C_EINCL**
Offset in comment section	**C_INFO**
Symbol table index	**C_FILE, C_BSTAT**
Offset relative to stack frame	**C_LSYM, C_PSYM**
Register number	**C_RPSYM, C_RSYM**
Offset within common block	**C_ECOML**

`n_scnum` Specifies a section number associated with one of the following symbols:

-2	Specifies **N_DEBUG**, a special symbolic debugging symbol.
-1	Specifies **N_ABS**, an absolute symbol. The symbol has a value but is not relocatable.
0	Specifies **N_UNDEF**, an undefined external symbol.
Any other value	Specifies the section number where the symbol was defined.

`n_type` Used in COFF for type information. This use is obsolete in XCOFF. For **C_EXT** and **C_HIDEXT** symbols, this field should contain `0x0020` for function symbols and `0` otherwise. This field has a special purpose for **C_FILE** symbols. See "File Auxiliary Entry for the **C_FILE** Symbol" for more information.

`n_sclass` Specifies the storage class of the symbol. The **storclass.h** and **dbxstclass.h** files contain the definitions of the storage classes. See "Symbol Table Field Contents by Storage Class" for more information.

`n_numaux` Specifies the number of auxiliary entries for the symbol. If more than one auxiliary entry is required for a symbol, the order of the auxiliary entries is determined by convention. That is, no flag field in the auxiliary entries can be used to distinguish one type of auxiliary entry from another.

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

Symbol Table Auxiliary Information

The symbol table contains auxiliary entries to provide supplemental information for a symbol. The auxiliary entries for a symbol follow its symbol table entry. The length of each auxiliary entry is the same as a symbol table entry (18 bytes). The format and quantity of auxiliary entries depend on the storage class (`n_sclass`) and type (`n_type`) of the symbol table entry.

In XCOFF32, symbols having a storage class of **C_EXT** or **C_HIDEXT** and more than one auxiliary entry must have the csect auxiliary entry as the last auxiliary entry. In XCOFF64, the `x_auxtype` field of each auxiliary symbol table entry differentiates the symbols, but the convention is to generate the csect auxiliary symbol table entry last.

File Auxiliary Entry for C_FILE Symbols

The file auxiliary symbol table entry is defined to contain the source file name and compiler-related strings. A file auxiliary entry is optional and is used with a symbol table entry that has a storage-class value of **C_FILE**. The C language structure for a file auxiliary entry can be found in the **x_file** structure in the **syms.h** file.

The **C_FILE** symbol provides source file-name information, source-language ID and CPU-version ID information, and, optionally, compiler-version and time-stamp information.

The `n_type` field of the symbol table entry identifies the source language of the source file and the CPU version ID of the compiled object file. The field information is as follows:

`Source Language ID`	Overlays the high-order byte of the `n_type` field. This field contains the source-language identifier. The values for this field are defined in the `e_lang` field in "Exception Section" . This field can be used by the symbolic debuggers to determine the source language. The optional values for this field are 248 (TB_OBJECT) for symbols from object files with no **C_FILE** symbol table entry; or 249 (TB_FRONT) or 250 (TB_BACK) for generated entries used to provide debugging information. If the source language is TB_FRONT or TB_BACK, the 8-character name field begins with `' '` (blank) , '\0'(NULLl). If the source language is TB_FRONT, the third byte is the stabstring compaction level for the object file, and the n_offset field contains the symbol table index of the TB_BACK symbol table entry, if it exists, or 0 otherwise.

`CPU Version ID` Defined as the low-order byte of the `n_type` field. Decribes the kind of instructions generated for the file. The following values are defined:

0	Reserved.
1	Specifies , 32-bit mode.
2	Reserved.
3	Specifies the common intersection of 32-bit and Processor.
4	Specifies Processor.
5	Specifies any mix of instructions between different architectures.
6	Specifies a mix of and instructions ().
7-223	Reserved.
224	Specifies instructions.
225-255	Reserved.

If both fields are 0, no information is provided about the source language.

File Name Auxiliary Entry Format			
Offset	**Length in Bytes**	**Name**	**Description**
0	14	`x_fname`	Source file string
0	4	`x_zeroes`	Zero, indicating file string in string table (overlays first 4 bytes of `x_fname`)
4	4	`x_offset`	Offset of file string in string table (overlays 5th-8th bytes of `x_fname`)
14	1	`x_ftype`	File string type
15	2		Reserved. Must contain 0.
17	1	`x_auxtype`	Auxiliary symbol type(XCOFF64 only)

Field Definitions

The following defines the fields listed above:

`x_fname`	Specifies the source file name or compiler-related string. If the file name or string is longer than 8 bytes, the field is interpreted as the following two fields: `x_zeroes` A value of 0 indicates that the source file string is in the string table (overlays first 4 bytes of `x_fname`). `x_offset` Specifies the offset from the beginning of the string table to the first byte of the source file string (overlays last 4 bytes of `x_fname`).
`x_ftype`	Specifies the source-file string type. **0 XFT_FN** Specifies the source-file name **1 XFT_CT** Specifies the compiler time stamp **2 XFT_CV** Specifies the compiler version number **128 XFT_CD** Specifies compiler-defined information
(no name)	Reserved. This field must contain 2 bytes of 0.
`x_auxtype`	(XCOFF64 only) Specifies the type of auxiliary entry. Contains _AUX_FILE for this auxiliary entry.

If the file auxiliary entry is not used, the symbol name is the name of the source file. If the file auxiliary entry is used, then the symbol name should be `.file`, and the first file auxiliary entry (by convention) contains the source file name. More than one file auxiliary entry is permitted for a given symbol table entry. The `n_numaux` field contains the number of file auxiliary entries.

csect Auxiliary Entry for C_EXT and C_HIDEXT Symbols

The csect auxiliary entry identifies csects (section definitions), entry points (label definitions), and external references (label declarations). A csect auxiliary entry is required for each symbol table entry that has a storage class value of **C_EXT** or **C_HIDEXT**. See "Symbol Table Entry (syms.h)" for more information. By convention, the csect auxiliary entry in an XCOFF32 file must be the last auxiliary entry for any external symbol that has more than one auxiliary entry. The C language structure for a csect auxiliary entry can be found in the **x_csect** structure in the **syms.h** file.

csect Auxiliary Entry Format					
XCOFF32		XCOFF64		Name	Description
Offset	Length	Offset	Length		
0	4	N/A		`x_scnlen`	(See field definition section)
N/A		0	4	`x_scnlen_lo`	(See field definition section) Low 4 bytes of section length
4	4	4	4	`x_parmhash`	Offset of parameter type-check hash in `.typchk` section
8	2	8	2	`x_snhash`	`.typchk` section number
10	1	10	1	`x_smtyp`	Symbol alignment and type 3-bit symbol alignment (log 2) 3-bit symbol type
11	1	11	1	`x_smclas`	Storage mapping class
12	4	N/A		`x_stab`	Reserved
16	2	N/A		`x_snstab`	Reserved
N/A		12	4	`x_scnlen_hi`	(See field definition section) High 4 bytes of section length
N/A		16	1	(pad)	Reserved
N/A		17	1	`x_auxtype`	Contains _AUX_CSECT; indicates type of auxiliary entry

Field Definitions

The following defines the fields listed above:

`x_scnlen`
: Specifies a meaning dependent on `x_smtyp` as follows:

If	Then
XTY_SD	`x_scnlen` contains the csect length.
XTY_LD	`x_scnlen` contains the symbol table index of the containing csect.
XTY_CM	`x_scnlen` contains the csect length.
XTY_ER	`x_scnlen` contains 0.

In the XCOFF64 format, the value of `x_scnlen` is divided into two fields: `x_scnlen_hi`, representing the upper 4 bytes of the value, and `x_scnlen_lo`, representing the lower 4 bytes of the value.

`x_parmhash`
: Specifies the byte offset of the parameter type-check string in the `.typchk` section. The byte offset is from the beginning of the `.typchk` section in an XCOFF file. The byte offset points to the first byte of the parameter type-check string (not to its length field). See "Type-Check Section" for more information. A value of 0 in the `x_parmhash` field indicates that the parameter type-checking string is not present for this symbol, and the symbol will be treated as having a universal hash. The value should be 0 for **C_HIDEXT** symbols.

`x_snhash`
: Specifies the `.typchk` section number. The XCOFF section number containing the parameter type-checking strings. The section numbers are one-based. For compatibility with object files generated by some compilers, if `x_parmhash` is not equal to 0 but `x_snhash` does equal 0, then the first `.typchk` section in the file is used. The value should be 0 for **C_HIDEXT** symbols.

`x_smtyp`
: Specifies symbol alignment and type:

Bits 0-4
: Contains a 5-bit csect address alignment value (log base 2). For example, a value of 3 in this field indicates 23, or 8, meaning the csect is to be aligned on an 8-byte address value. The alignment value is used only when the value of bits 5-7 of the `x_smtyp` field is either **XTY_SD** or **XTY_CM**.

Bits 5-7
: Contains a 3-bit symbol type field. See the definitions for bits 5-7 of the `l_smtype` field in "Loader Section" for more information.

`x_smclas`
Specifies the csect storage-mapping class. This field permits the binder to arrange csects by their storage-mapping class. The `x_smclas` field is used only when the value of bits 5-7 of the `x_smtyp` field is either **XTY_SD** or **XTY_CM**.

The following storage-mapping classes are read-only and normally mapped to the `.text` section:

Value Class — **Description**

0 XMC_PR
Specifies program code. The csect contains the executable instructions of the program.

1 XMC_RO
Specifies a read-only constant. The csect contains data that is constant and will not change during execution of the program.

2 XMC_DB
Specifies the debug dictionary table. The csect contains symbolic-debugging data or exception-processing data. This storage mapping class was defined to permit compilers with special symbolic-debugging or exception-processing requirements to place data in csects that are loaded at execution time but that can be collected separately from the executable code of the program.

6 XMC_GL
Specifies global linkage. The csect provides the interface code necessary to handle csect relative calls to a target symbol that can be out-of-module. This global linkage csect has the same name as the target symbol and becomes the local target of the relative calls. As a result, the csect maintains position-independent code within the `.text` section of the executable XCOFF object file.

7 XMC_XO
Specifies extended operation. A csect of this type has no dependency on (references through) the TOC. It is intended to reside at a fixed address in memory such that it can be the target of a branch-absolute instruction.

12 XMC_TI
Reserved.

13 XMC_TB
Reserved.

The following storage-mapping classes are read/write and normally mapped to the `.data` or `.bss` section:

Value Class — **Description**

5 XMC_RW
Specifies read/write data. A csect of this type contains initialized or uninitialized data that is permitted to be modified during program execution. If the `x_smtyp` value is **XTY_SD**, the csect contains initialized data and is mapped into the `.data` section. If the `x_smtyp` value is **XTY_CM**, the csect is uninitialized and is mapped into the `.bss` section. Typically, all the initialized static data from a C source file is contained in a single csect of this type. The csect would have a storage class value of **C_HIDEXT**. An initialized definition for a global data scalar or structure from a C source file is contained in its own csect of this type. The csect would have a storage class value of **C_EXT**. A csect of this type is accessible by name references from other object files.

15 XMC_TC0
Specifies TOC anchor for TOC addressability. This is a zero-length csect whose `n_value` address provides the base address for TOC relative addressability. Only one csect of type **XMC_TC0** is permitted per section of an XCOFF object file. In implementations that permit compilers and assemblers to generate multiple `.data` sections, there must be a csect of type **XMC_TC0** in each section that contains data that is referenced (by way of a relocation entry) as a TOC-relative data item. Some hardware architectures limit the value that a relative displacement field within a load instruction may contain. This limit then becomes an inherent limit on the size of a TOC for an executable XCOFF object. For and RS/6000 , this limit is 65,536 bytes, or 16,384 4-byte TOC entries.

3 XMC_TC
Specifies general TOC entry. A csect of this type is usually 4 bytes in length and contains the address of another csect or global symbol. This csect provides addressability to other csects or symbols. The symbols may be contained in either the local executable XCOFF object or in another executable XCOFF object. Special processing semantics are used by the binder to eliminate duplicate TOC entries as follows:

- Symbols that have a storage class value of **C_EXT** are global symbols and must have names (a non-null `n_name` field). These symbols require no special TOC processing logic to combine duplicate entries. Duplicate entries with the same `n_name` value are combined into a single entry.
- Symbols that have a storage class value of **C_HIDEXT** are not global symbols, and duplicate entries are resolved by context. Any two such symbols will be defined as duplicates and combined into a single entry whenever the following conditions are met:
  - The `n_name` fields are the same. That is, they have either a null name or the same name string.
  - Each is 4 bytes long.
  - Each has a single RLD entry that references external symbols with the same name.

To minimize the number of duplicate TOC entries that cannot be combined by the binder, compilers and assemblers should adhere to a common naming convention for TOC entries. By convention, compilers and assemblers produce TOC entries that have a storage class value of **C_HIDEXT** and an `n_name` string that is the same as the `n_name` value for the symbol that the TOC entry addresses.

16 XMC_TD
Specifies scalar data entry in the TOC. A csect that is a special form of an **XMC_RW** csect that is directly accessed from the TOC by compiler generated code. This lets some frequently used globol symbols be accessed directly from the TOC rather than indirectly through an address pointer csect contained in the TOC. A csect of type **XMC_TD** has the following characteristics:

- The compiler generates code that is TOC relative to directly access the data contained in the csect of type **XMC_TD**.
- It is 4-bytes long or less.
- It has initialized data that can be modified as the program runs.
- If a same named csect of type **XMC_RW** or **XMC_UA** exist, it is replaced by the **XMC_TD** csect.

For the cases where TOC scalar cannot reside in the TOC, the binder must be capable of transforming the compiler generated TOC relative instruction into a conventional indirect addressing instruction sequence. This transformation is necessary if the TOC scalar is contained in a shared object.

10 XMC_DS
Specifies a csect containing a function descriptor, which contains the following three values:

- The address of the executable code for a function.
- The address of the TOC anchor (TOC base address) of the module that contains the function.
- The environment pointer (used by languages such as Pascal and PL/I).

There is only one function descriptor csect for a function, and it must be contained within the same executable as the function itself is contained. The function descriptor has a storage class value of **C_EXT** and has an `n_name` value that is the same as the name of the function in the source file. The addresses of function descriptors are imported to and exported from an executable XCOFF file.

8 XMC_SV
Specifies 32-bit supervisor call descriptor csect. The supervisor call descriptors are contained within the operating system kernel. To an application program, the reference to a supervisor call descriptor is treated the same as a reference to a regular function descriptor. It is through the import/export mechanism that a function descriptor is treated as a supervisor call descriptor. These symbols are only available to 32-bit programs.

17 XMC_SV64
Specifies 64-bit supervisor call descriptor csect. See **XMV_SV** for supervisor call information. These symbols are only available to 64-bit programs.

18 XMC_SV3264
Specifies supervisor call descriptor csect for both 32-bit and 64-bit. See **XMV_SV** for supervisor call information. These symbols are available to both 32-bit and 64-bit programs.

4 XMC_UA
Unclassified. This csect is treated as read/write. This csect is frequently produced by an assembler or object file translator program that cannot determine the true classification of the resultant csect.

9 XMC_BS
Specifies BSS class (uninitialized static internal). A csect of this type is uninitialized, and is intended to be mapped into the `.bss` section. This type of csect must have a `x_smtyp` value of **XTY_CM**.

11 XMC_UC
Specifies unnamed FORTRAN common. A csect of this type is intended for an unnamed and uninitialized FORTRAN common. It is intended to be mapped into the `.bss` section. This type of csect must have a `x_smtyp` value of **XTY_CM**.

`x_stab`
Reserved (Unused for 64-bit).

`x_snstab`
Reserved (Unused for 64-bit).

Auxiliary Entries for the C_EXT and C_HIDEXT Symbols

Auxiliary symbol table entries are defined in XCOFF to contain reference and size information associated with a defined function. These auxiliary entries are produced by compilers and assembler for use by the symbolic debuggers. In XCOFF32, a function auxiliary symbol table entry contains the required information. In XCOFF64, both a function auxiliary entry and an exeption auxiliary entry may be needed. When both auxiliary entries are generated for a single **C_EXT** or **C_HIDEXT** symbol, the `x_size` and `x_endndx` fields must have the same values.

The function auxiliary symbol table entry is defined in the following table.

Function Auxiliary Entry Format					
XCOFF32		**XCOFF64**		**Name**	**Description**
Offset	**Length**	**Offset**	**Length**		
0	4	N/A		`x_exptr`	File offset to exception table entry.
4	4	8	4	`x_fsize`	Size of function in bytes
8	4	0	8	`x_lnnoptr`	File pointer to line number
12	4	12	4	`x_endndx`	Symbol table index of next entry beyond this function
16	1	16	1	(pad)	Unused
N/A		17	1	`x_auxtype`	Contains _AUX_FCN; Type of auxiliary entry

Field Definitions

The following defines the fields listed in the Function Auxiliary Entry Format table:

`x_exptr`
: (XCOFF32 only) This field is a file pointer to an exception table entry. The value is the byte offset from the beginning of the XCOFF object file. In an XCOFF64 file, the exception table offsets are in an exception auxiliary symbol table entry.

`x_fsize`
: Specifies the size of the function in bytes.

`x_lnnoptr`
: Specifies a file pointer to the line number. The value is the byte offset from the beginning of the XCOFF object file.

`x_endndx`
: Specifies the symbol table index of the next entry beyond this function.

The exception auxiliary symbol table entry, defined in XCOFF64 only, is shown in the following table.

Exception Auxiliary Entry Format (XCOFF64 only)			
Offset	**Length**	**Name**	**Description**
0	8	`x_exptr`	File offset to exception table entry.
8	4	`x_fsize`	Size of function in bytes
12	4	`x_endndx`	Symbol table index of next entry beyond this function
16	1	(pad)	Unused
17	1	`x_auxtype`	Contains _AUX_EXCEPT; Type of auxiliary entry

Field Definitions

The following defines the fields listed in the Exception Auxiliary Entry Format table:

`x_exptr` This field is a file pointer to an exception table entry. The value is the byte offset from the beginning of the XCOFF object file.

`x_fsize` Specifies the size of the function in bytes.

`x_endndx` Specifies the symbol table index of the next entry beyond this function.

Block Auxiliary Entry for the C_BLOCK and C_FCN Symbols

The section auxiliary symbol table entry is defined in XCOFF to provide information associated with the begin and end blocks of functions. The section auxiliary symbol table entry is produced by compilers for use by the symbolic debuggers.

Table Entry Format					
XCOFF32		**XCOFF64**			
Offset	**Length**	**Offset**	**Length**	**Name**	**Description**
0	4	N/A		(no name)	Reserved
4	2	0	4	`x_lnno`	Source line number
6	12	4	13	(no name)	Reserved
N/A		17	1	`x_auxtype`	Contains _AUX_SYM; Type of auxiliary entry

Field Definitions

The following defines the fields above:

(no name)	Reserved.
x_lnno	Specifies the line number of a source file. The maximum value of this field is 65535 for XCOFF64 and 2^{32} for XCOFF64.
(no name)	Reserved.

Section Auxiliary Entry for the C_STAT Symbol

The section auxiliary symbol table entry ID is defined in XCOFF32 to provide information in the symbol table concerning the size of sections produced by a compiler or assembler. The generation of this information by a compiler is optional, and is ignored and removed by the binder.

Section Auxiliary Entry Format (XCOFF32 Only)			
Offset	**Length in Bytes**	**Name**	**Description**
0	4	x_scnlen	Section length
4	2	x_nreloc	Number of relocation entries
6	2	x_nlinno	Number of line numbers
8	10	(no name)	Reserved

Field Definitions

The following list defines the fields:

x_scnlen	Specifies section length in bytes.
x_nreloc	Specifies the number of relocation entries. The maximum value of this field is 65535.
x_nlinno	Specifies the number of line numbers. The maximum value of this field is 65535.
(no name)	Reserved.

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format." For more information on the symbol table, see "Symbol Table Information."

For information on debugging, see "Debug Section."

Symbol Table Field Contents by Storage Class

This section defines the symbol table field contents for each of the defined storage classes (n_sclass) that are used in XCOFF. The following table lists storage class entries in alphabetic order. See "Symbol Table Entry (syms.h)" for more information.

Symbol Table by Storage Class				
Class Definition	**n_name**	**n_value**	**n_scnum**	**Aux. Entry**

C_BCOMM 135 Beginning of common block	Name of the common block*	0, undefined	**N_DEBUG**	
C_BINCL 108 Beginning of include file	Source name of the include file**	File pointer	**N_DEBUG**	
C_BLOCK 100 Beginning or end of inner block	`.bb` or `.eb`	Relocatable address	**N_SCNUM**	BLOCK
C_BSTAT 143 Beginning of static block	`.bs`	Symbol table index	**N_DEBUG**	
C_DECL 140 Declaration of object (type)	Debugger stabstring*	0, undefined	**N_SCNUM**	
C_ECOML 136 Local member of common block	Debugger stabstring*	Offset within common block	**N_ABS**	
C_ECOMM 137 End of common block	Debugger stabstring*	0, undefined	**N_DEBUG**	
C_EINCL 109 End of include file	Source name of the include file**	File pointer	**N_DEBUG**	
C_ENTRY 141 Alternate entry	*	0, undefined	**N_DEBUG**	
C_ESTAT 144 End of static block	`.es`	0, undefined	**N_DEBUG**	
C_EXT 2 External symbol (defining external symbols for binder processing)	Symbol Name**	Relocatable address	**N_SCNUM** or **N_UNDEF**	FUNCTION CSECT
C_FCN 101 Beginning or end of function	`.bf` or `.ef`	Relocatable address	**N_SCNUM**	BLOCK
C_FILE 103 Source file name and compiler information	`.file` or source file name (if no auxiliary entries)**	Symbol table index	**N_DEBUG**	FILE
C_FUN 142 Function or procedure	Debugger stabstring*	Offset within containing csect	**N_ABS**	
C_GSYM 128 Global variable	Debugger stabstring*	0, undefined	**N_DEBUG**	
C_HIDEXT 107 Unnamed external symbol	Symbol Name or null**	Relocatable address	**N_SCNUM**	FUNCTION CSECT

C_INFO 100 Comment section reference	Info Name Identifier or null**	Offset within comment section	**N_SCNUM**	
C_LSYM 129 Automatic variable allocated on stack	Debugger stabstring*	Offset relative to stack frame	**N_ABS**	
C_NULL 0 Symbol table entry marked for deletion.		0x00DE1E00		Any
C_PSYM 130 Argument to subroutine allocated on stack	Debugger stabstring*	Offset relative to stack frame	**N_ABS**	
C_RPSYM 132 Argument to function or procedure stored in register	Debugger stabstring*	Register number	**N_ABS**	
C_RSYM 131 Register variable	Debugger stabstring*	Register number	**N_ABS**	
C_STAT 3 Static symbol (Unknown. Some compilers generate these symbols in the symbol table to identify size of the `.text` , `.data` , and `.bss` sections. Not used or preserved by binder.)	Symbol Name**	Relocatable address	**N_SCNUM**	SECTION
C_STSYM 133 Statically allocated symbol	Debugger stabstring*	Offset within csect	**N_ABS**	
C_TCSYM 134 Reserved	Debugger stabstring*			

Notes:

1. *For long name, the `n_offset` value is an offset into the `.debug` section.
2. **For long name, the `n_offset` value is an offset into the string table.

Storage Classes by Usage and Symbol Value Classification

Following are the storage classes used and relocated by the binder. The symbol values (`n_value`) are addresses.

Class	Description
C_EXT	Specifies an external or global symbol
C_HIDEXT	Specifies an internal symbol
C_BLOCK	Specifies the beginning or end of an inner block (`.bb` or `.eb`)
C_FCN	Specifies the beginning or end of a function (`.bf` or `.ef` only)
C_STAT	Specifies a static symbol (contained in statics csect)

Following are storage classes used by the binder and symbolic debugger or by other utilities for file scoping and accessing purposes:

C_FILE Specifies 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.

C_BINCL Specifies the beginning of include header file. The `n_value` field is the line number byte offset in the object file to the first line number from the include file.

C_EINCL Specifies the end of include header file. The `n_value` field is the line number byte offset in the object file to last line number from the include file.

C_INFO Specifies the location of a string in the comment section. The `n_value` field is the offset to a string of bytes in the specified **STYP_INFO** section. The string is preceded by a 4-byte `length` field. The `n_name` field is preserved by the binder. An application-defined unique name in this field can be used to filter access to only those comment section strings intended for the application.

Following are the storage classes that exist only for symbolic debugging purposes:

C_BCOMM Specifies the beginning of a common block. The `n_value` field is meaningless; the name is the name of the common block.

C_ECOML Specifies a local member of a common block. The `n_value` field is byte-offset within the common block.

C_ECOMM Specifies the end of a common block. The `n_value` field is meaningless.

C_BSTAT Specifies the beginning of a static block. The `n_value` field is the symbol table index of the csect containing static symbols; the name is .bs.

C_ESTAT Specifies the end of a static block. The `n_value` field is meaningless; the name is .es.

C_DECL Specifies a declaration of object (type declarations). The `n_value` field is undefined.

C_ENTRY Specifies an alternate entry (FORTRAN) and has a corresponding **C_EXT** symbol. The `n_value` field is undefined.

C_FUN Specifies a function or procedure. May have a corresponding **C_EXT** symbol. The `n_value` field is byte-offset within the containing csect.

C_GSYM Specifies a global variable and has a corresponding **C_EXT** symbol. The `n_value` field is undefined.

C_LSYM Specifies an automatic variable allocated on the stack. The `n_value` field is byte offset relative to the stack frame (platform dependent).

C_PSYM Specifies an argument to a subroutine allocated on the stack. The `n_value` field is byte-offset relative to the stack frame (platform dependent).

C_RSYM Specifies a register variable. The `n_value` field is the register number.

C_RPSYM Specifies an argument to a function or procedure stored in a register. The `n_value` field is the register number where argument is stored.

C_STSYM Specifies a statically allocated symbol. The `n_value` field is byte-offset within csect pointed to by containing **C_BSTAT** entry.

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format." For more information on the symbol table, see "Symbol Table Information."

For information on debugging, see "Debug Section."

String Table

IN XCOFF32, the string table contains the names of symbols that are longer than 8 bytes. In XCOFF64, the string table contains the names of all symbols. If the string table is present, the first 4 bytes contain the length (in bytes) of the string table, including the length of this length field. The remainder of the table is a sequence of null-terminated ASCII strings. If the `n_zeroes` field in the symbol table entry is 0, then the `n_offset` field gives the byte offset into the string table of the name of the symbol.

If a string table is not used, it may be omitted entirely, or a string table consisting of only the length field (containing a value of 0 or 4) may be used. A value of 4 is preferable. The following table shows string table organization.

String Table Organization		
Offset	**Length in Bytes**	**Description**
0	4	Length of string table.
4	*n*	Symbol name string, null-terminated.
		Field repeats for each symbol name.

For general information on the XCOFF file format, see "XCOFF Object (a.out) File Format."

dbx Stabstrings

The debug section contains the symbolic debugger stabstrings (symbol table strings). It is generated by the compilers and assemblers. It provides symbol attribute information for use by the symbolic debugger.

See "Debug Section" for a general discussion.

Stabstring Terminal Symbols

In the stabstring grammar, there are five types of terminal symbols, which are written in all capital letters. These symbols are described by the regular expressions in the following list:

> **Note:** The [ ] (brackets) denote one instance, [ ]* (brackets asterisk) denote zero or more instances, [ ]+ (brackets plus sign) denote one or more instances, ( ) (parentheses) denote zero or one instance, .* (dot asterisk) denotes a sequence of zero or more bytes, and | (pipe) denotes alternatives.

Symbol	Regular Expression
NAME	[^ **; : ' "**] (A name consists of any non-empty set of characters, excluding **; : '** or ".)
STRING	**'.*'** | ".*", where \", \', or \\ can be used inside the string Within a string, the \ (backslash character) may have a special meaning. If the character following the \ is another \, one of the backslashes is ignored. If the next character is the quote character used for the current string, the string is interpreted as containing an embedded quote. Otherwise, the \ is interpreted literally. However, if the closing quote is the last character in the stabstring, and a \ occurs immediately before the quote, the \ is interpreted literally. This use is not recommended. The \ must be quoted only in the following instances: • The \ is the last character in the string (to avoid having the closing quote escaped). • The \ is followed by the current quote character. • The \ is followed by another \. An escaped quote is required only when a single string contains both a single quote and a double quote. Otherwise, the string should be quoted with the quote character not contained in the strings. A string can contain embedded null characters, so utilities that process stabstrings must use the length field to determine the length of a stabstring.
INTEGER	(-)[**0-9**]+
HEXINTEGER	[**0-9A-F**]+ The hexadecimal digits **A-F** must be uppercase.
REAL	[+-][**0-9**]+(**.**)[**0-9**]*([**eEqQ**](+-)[**0-9**]+) | (+-)**INF** | **QNAN** | **SNAN** Real numbers are the same strings recognized by the **scanf** subroutine when using the **"%lf"** pattern. Therefore, white space may occur before a real number.

Stabstring Grammar

REALs may be preceded by white space, and STRINGs may contain any characters, including null and blank characters. Otherwise, there are no null or blank characters in a stabstring.

Long stabstrings can be split across multiple symbol table entries for easier handling. In the stabstring grammar, a **#** (pound sign) indicates a point at which a stabstring may be continued. A continuation is indicated by using either the ? (question mark) or \ as the last character in the string. The next part of the stabstring is in the name of the next symbol table entry. If an alternative for a production is empty, the grammar shows the keyword /*EMPTY*/.

The following list contains the stabstring grammar:

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
	C REAL**,** REAL	Complex constant
	S *TypeID* **,** *NumElements* **,** *NumBits* **,** *BitPattern*	Set constant.
OrdValue:	Associated numeric value: INTEGER	
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), excluding those that are valid for **T** *TypeID*
	T *TypeID*	Struct, union, class, 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 from 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)
Y	FORTRAN pointer variable
Z *TypeID* NAME	FORTRAN pointee variable

Label: Label:

L	Label name.

Proc: Different types of functions and procedures:

f *TypeID*	Private function of type TypeID
g *TypeID*	Generic function (FORTRAN)
m *TypeID*	Module (Modula-2, ext. Pascal)
J *TypeID*	Internal function of type *TypeID*
F *TypeID*	External function of type *TypeID*
I	(capital 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

TypeAttrs: **@** *TypeAttrList* **;**

Note: Type attributes (*TypeAttrs)* are extra information associated with a type, such as alignment constraints or pointer-checking semantics. The **dbx** program 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**.

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

TypeDef:

Basic descriptions of objects:

INTEGER	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 *EnumSpec* **;**	Enumerated type (default size, 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*
k *TypeID*	C++ constant type
l ; #	Usage-is-index; specific to COBOL
m *OptVBaseSpec OptMultiBaseSpec TypeID* **:** *TypeID* **:** *TypeID* **;**	
	C++ pointer to member type; the first *TypeID* is the member type; the second is the type of the class
n *TypeID* **; #** *NumBytes*	String type, with maximum string length indicated by *NumBytes*
o NAME **;**	Opaque type
o NAME **,** *TypeID*	Opaque type with definition of *TypeID*
w *TypeID*	Wide character
z *TypeID* **; #** *NumBytes*	Pascal gstring type
C *Usage*	COBOL Picture
I *NumBytes* **; #** *PicSize*	(uppercase i) Index is type; specific to COBOL
K *CobolFileDesc;*	COBOL File Descriptor
M *TypeID* **; #** Bound	Multiple instance type of *TypeID* with length indicated by Bound
N	Pascal *Stringptr*
S *TypeID*	Set of type *TypeID*
***** *TypeID*	Pointer of type *TypeID*
& *TypeID*	C++ reference type
V *TypeID*	C++ volatile type
Z	C++ ellipses parameter type

Array
Subrange
ProcedureType

For function types rather than declarations

Record	Record, structure, union, or group types

EnumSpec:

List of enumerated scalars:

EnumList	Enumerated type (C and other languages)
TypeID : EnumList	C++ enumerated type with repeating integer type

EnumList:
Enum

EnumList Enum

Enum:

Enumerated scalar description:
NAME **:** *OrdValue* **, #**

Array:	Array descriptions:	
	a *TypeID* **; #** *TypeID*	Array; *FirstTypeID* is the index type
	A *TypeID*	Open array of *TypeID*
	D INTEGER **,** *TypeID*	N-dimensional dynamic array of *TypeID*
	E INTEGER **,** *TypeID*	N-dimensional dynamic subarray of *TypeID*
	O INTEGER **,** *TypeID*	New open array
	P *TypeID* **; #** *TypeID*	Packed array
Subrange:	Subrange descriptions:	
	r *TypeID* **; #** *Bound* **; #** *Bound*	
		Subrange type (for example, 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
	S	Bound passed by value in static storage
	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.	
TParamList:	Types of parameters in Modula-2 function variable:	
	TParam	Type of parameter and passing method
	TParamList TParam	
TParam:	Type and passing method	
	TypeID **,** *PassBy* **; #**	
NamedTParamList:	Types of parameters in Pascal-routine variable: /*EMPTY*/ *NamedTPList*	

NamedTPList:	NamedTParam *NamedTPList NamedTParam*
NamedTParam:	Named type and passing method: *Name* **:** *TypeID* **,** *PassBy InitBody* **; #** : *TypeID* **,** *PassBy InitBody* **; #** Unnamed parameter
Record:	Types of structure declarations: **s** *NumBytes* # *FieldList* **;** Structure or record definition **u** *NumBytes* **#** *FieldList* **;** Union **v** *NumBytes* # *FieldList VariantPart* **;** Variant Record **Y** *NumBytes ClassKey OptPBV OptBaseSpecList* **(** *ExtendedFieldListOptNameResolutionList* **;** C++ class **G** *Redefinition* **, n** *NumBits* # *FieldList* **;** COBOL group without conditionals **Gn** *NumBits FieldList* **;** **G** *Redefinition* **, c** *NumBits* # *CondFieldList* **;** COBOL group with conditionals **Gc** *NumBits CondFieldList* **;**
OptVBaseSpec:	**v** ptr-to-mem class has virtual bases. /*EMPTY*/ Class has no virtual bases.
OptMultiBaseSpec:	**m** Class is multi-based. /*EMPTY*/ Class is not multi-based.
OptPBV:	**V** Class is always passed by value. /*EMPTY*/ Class is never passed by value.
ClassKey:	**s** struct **u** union **c** class
OptBaseSpecList:	/*EMPTY*/ *BaseSpecList*
BaseSpecList:	*BaseSpec* *BaseSpecList* **,** *BaseSpec*
BaseSpec:	*VirtualAccessSpec BaseClassOffset* **:** *ClassTypeID*
BaseClassOffset:	INTEGER Base record offset in bytes
ClassTypeID:	*TypeID* Base class type identifier
VirtualAccessSpec:	**v** *AccessSpec* Virtual **v** Virtual *AccessSpec* /*EMPTY*/

GenSpec:	**c**	Compiler-generated
	/*EMPTY*/	
AccessSpec:	**i #**	Private
	o #	Protected
	u #	Public
AnonSpec:	**a**	Anonymous union member
	/*EMPTY*/	
VirtualSpec:	**v p**	Pure virtual
	v	Virtual
	/*EMPTY*/	

ExtendedFieldList: *ExtendedFieldList ExtendedField*
/*EMPTY*/

ExtendedField: *GenSpec AccessSpec AnonSpec DataMember*
GenSpec VirtualSpec AccessSpec OptVirtualFuncIndex MemberFunction
AccessSpec AnonSpec NestedClass
AnonSpec FriendClass
AnonSpec FriendFunction

DataMember: *MemberAttrs* **:** *Field* **;**

MemberAttrs: *IsStatic IsVtblPtr IsVBasePtr*

IsStatic:	/*EMPTY*/	
	s	Member is static.
IsVtblPtr:	/*EMPTY*/	
	p INTEGER NAME	Member is vtbl pointer; NAME is the external name of v-table.
IsVBasePtr:	/*EMPTY*/	
	b	Member is vbase pointer.
	r	Member is vbase self-pointer.

Member Function: **[** *FuncType MemberFuncAttrs* **:** NAME **:** *TypeID* **; #**

MemberFuncAttrs: *IsStatic*
IsInline
IsConst
IsVolatile

IsInline:	/*EMPTY*/	
	i	Inline function
IsConst:	/*EMPTY*/	
	k	const member function
IsVolatile:	/*EMPTY*/	
	V	Volatile member function

NestedClass: **N** *TypeID* **; #**

FriendClass: **(** *TypeID* **; #**

FriendFunction: **]** NAME **:** *TypeID* **; #**

OptVirtualFuncIndex: /*EMPTY*/
INTEGER

FuncType:

f	Member function
c	Constructor
d	Destructor

InitBody: STRING
/*EMPTY*/

OptNameResolutionList: /*EMPTY*/
) *NameResolutionList*

NameResolutionList: *NameResolution* **,** *NameResolutionList*
NameResolution

NameResolution: Name is resolved by compiler.
MemberName **:** *ClassTypeID*

MemberName **:**	Name is ambiguous.

MemberName: NAME

FieldList: Structure content descriptions:

Field	/*EMPTY*/
FieldList Field	Member of record or union.

Field: Structure-member type description:

NAME **:** *TypeID* **,** *BitOffset* **,** *NumBits* **; #**

VariantPart: Variant portion of variant record:

[*Vtag VFieldList*]	Variant description

VTag: Variant record tag:

(*Field* Member of variant record

(NAME **: ; #**	Variant key name

VFieldList: Variant record content descriptions:

VList *VFieldList VList*	Member of variant record
VList:	Variant record fields:
VField *VField VariantPart*	Member of variant record

VField: Variant record member type description:

(*VRangeList* **:** *FieldList*	Variant with field list

VRangeList: List of variant field labels:

VRange *VRangeList* **,** *VRange*	Member of variant record

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

CondFieldList: Conditions,#FieldList
FieldList# ;

Conditions: /*Empty*/
Conditions condition

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* **;**

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*
ConditionPrimitive:	Primitive type of Condition:

n *Sign DecimalSite*	Numeric conditional
a	Alphanumeric conditional
f	Figurative conditional

Sign:	For types with explicit sign:

+	Positive
-	Negative
[^+-]	Not specified

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 **:** *ArbitraryCharacters #*	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

PassBy:	Parameter passing method:

INTEGER	0 = passed-by reference; 1 = passed-by value

Related Information

Header Files.

The **as** command, **dbx** command, **dump** command, **ld** command, **size** command, **strip** command, and **what** command.

Chapter 3. Special Files

A *special file* is associated with a particular hardware device or other resource of the computer system. The operating system 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, in that they:

- Have path names that appear in a directory.
- Have the same access protection as ordinary files.
- 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 the disk drive
- A pseudo device, such as the physical memory of the computer (**/dev/mem**) or the null file (**/dev/null**).

Special files are distinguished from other files by having a file type (`c` or `b`, for character or block) stored in the 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.

> **Attention:** Data corruption, loss of data, or loss of system integrity (a system crash) 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 the operating system and are solely for system use in managing file systems, paging devices, and logical volumes. These files should not be used for other purposes.

Several special files are provided with the operating system. By convention, special files are located in the **/dev** directory.

Related Information

File Formats Overview defines and describes file formats.

Header Files Overview

3270cn Special File

Purpose

Provides access to 3270 connection adapters by way of the 3270 connection adapter device handler.

Description

The **3270c***n* character special file provides access to the 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:

Distributed Function Terminal (DFT) mode

In DFT mode, the adapter can appear as multiple SNA or non-SNA display sessions, non-SNA 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 3278/79 display stations. If the attached control unit does not support Extended Asynchronous Event Device Status, either the control unit port 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 3278/79 display station.

The device handler supports up to four 3270 connection adapters, each of which may have up to five DFT sessions or one CUT session.

The **/usr/include/sys/io3270.h** file contains the definitions of the structures used by the device handler.

Usage Considerations

When accessing the 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 **open** and **close** 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** (non-SNA DFT mode only)
- **write**
- **writex** (non-SNA DFT mode only)
- **ioctl**

open and close Subroutines

The device handler supports the **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 precede 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 the * (asterisk) is defined by *P*/*a*, as follows:

P/00, *P*/01, *P*/02,...*P*/1F
The printer session specified by the *P* variable is equal to the control unit session address, and the value of *a* is less than or equal to 0x1F.

01 through 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.

read Subroutine in Non-SNA DFT Mode

Data received by the communication adapter from the host is placed in the buffer until the message is completed 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 operation.

If the **WDC_INQ** operation returns a status indicating that more data is available, the application should immediately issue another **read** call. Available data must be read as soon as possible to avoid degrading link or host performance.

If a **read** 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

read Subroutine in SNA DFT Mode

The communication adapter receives data from the control unit in individual SNA data segments. The device driver notifies the application that data is available. During the **read** subroutine call, the data is transferred to the application’s user space from the device driver’s kernel space (without the TCA header from the control unit), and control is passed back to the application. The device driver acknowledges each SNA data segment received, making it unnecessary for the application to inquire about the link status after the **read** call.

Note: The **STAT_ACK** ioctl operation is not valid in SNA DFT mode.

Unlike non-SNA DFT mode, neither chaining nor command interpretation is performed by the device driver in SNA DFT mode. The application must both accumulate SNA data segments to form an response unit (RU) and interpret any 3270 data contained within.

readx Subroutine in Non-SNA 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** call, 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 Additional data is available for this link address.

WDI_COMM A communication error occurred. The `io_status` field contains the corresponding message code.

WDI_PROG A program error occurred. The `io_status` field contains the corresponding message code.

WDI_MACH A hardware error occurred. The `io_status` field contains the corresponding message code.

WDI_FATAL An error occurred that prevents further communication with the host. This flag is optionally set in addition to the **WDI_COMM**, **WDI_PROG**, or **WDI_MACH** flag. It is also set when a coax failure occurs. In this case, the `io_status` field contains a value of **WEB_610**, but the **WDI_COMM**, **WDI_PROG**, or **WDI_MACH** flag is 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 more data is available, another **readx** subroutine should be issued immediately. Available data must be read as soon as possible to avoid degrading 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 is sent using the **WDC_SSTAT** ioctl operation.

write Subroutine in Non-SNA DFT Mode

In non-SNA DFT mode, the **write** subroutine sends an inbound 3270 data stream to the host. The buffer specified on a **write** subroutine call must contain a complete inbound 3270 data stream. The **write** call is complete when it has successfully transferred from the buffer specified on the subroutine call.

write Subroutine in SNA DFT Mode

In SNA DFT mode, the **write** subroutine transmits SNA data to the host system. This data can be either a 3270 data stream with SNA headers or an SNA response.

The application sends data to the device driver, one RU at a time. The device driver is then responsible for segmenting the inbound SNA data. If a second **write** call is made before the first call is processed, the second call does not proceed until the device driver is ready. After the data is transferred from the application's user space to the device driver's kernel space, the **write** subroutine completes and control is returned to the application.

If the device driver detects a coax disconnect between two **write** calls, the second **write** call will return to the application, with the **errno** global variable set to **EFAULT**.

writex Subroutine in Non-SNA DFT Mode

The **writex** subroutine sends an inbound 3270 data stream to the host. The buffer specified on a **writex** subroutine call must contain a complete inbound 3270 data stream.

The **write** subroutine is complete when it has successfully transferred 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 codes is set in the `io_flags` field of the **io3270** structure:

WDI_DAVAIL
Indicates that data is available for this link address; the data must be read before any write can occur.

WDI_COMM
Indicates a communication error. The `io_status` field contains the corresponding message code.

WDI_PROG
Indicates a program error. The `io_status` field contains the corresponding message code.

WDI_MACH
Indicates a hardware error. The `io_status` field contains the corresponding message code.

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** operations:

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
Valid only for non-SNA DFT mode. Provides the handler with the option to automatically acknowledge the receipt of a valid 3270 data stream. An acknowledgment is sent only if the beginning of the 3270 data stream consists of `0xF3 00 06 40 00 F1 C2 xx xx 10 14`, where the `xx` fields are not examined. This command also allows the driver not to indicate acknowledgment upon receipt of data.

WDC_INQ Queries the status of the last non-SNA **read** or **write** call issued by the application. Also, the WDC_INQ operation determines 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 is buffered either in the driver or in the communication adapter. The data should be read immediately to avoid its having an impact on performance.

In non-SNA DFT mode, a **write** or **writex** subroutine call cannot complete until the data has been read. In SNA DFT mode, the **WDI_DAVAIL** flag is used only to indicate that data is available when the device driver wakes up the application (if waiting on a **poll** or **select** call) after receiving data from the control unit.

WDI_COMM, **WDI_PROG**, or **WDI_MACH**
Indicates a communication check, program check, or machine check, respectively. In each of these cases, the `io_status` field contains a message code that specifies the type of check.

WDI_FATAL
Indicates that an error has occurred that prevents further communication between the application and the device driver, typically a coax disconnect or adapter failure. This flag may be set in conjunction with the **WDI_COMM**, **WDI_PROG**, or **WDI_MACH** flag. If the communications failure was caused by a coax disconnect, the `io_status` field contains a value of **WEB_610**.

WDI_WCUS_30
A communications check reminder that occurs when there is a network failure and the control unit is still communicating with the communication adapter. The specific type of error is contained in the `io_status` field as a `5XX` error code. The communications check reminder is cleared automatically after the network condition is corrected.

WDI_WCUS_31
Indicates that the communications check reminder has been cleared.

WDI_CU
Valid only for SNA DFT mode. Indicates that an **ACTLU** or **DACTLU** request was received by the device driver. The accompanying data is contained in the `io_extra` field of the **io3270** structure.

WDC_POR The link address is first disabled and then re-enabled to emulate a 3270 terminal power-on reset function.

WDC_SSTAT Valid only for non-SNA DFT mode. 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
Received a bad buffer order or an invalid buffer address.

STAT_UNSUP
Received an unsupported 3270 command.

The **/usr/include/sys/io3270.h** 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 **3270cn** special file:

EBUSY An open was requested for a channel that is already open.

EFAULT A buffer specified by the caller was not valid.

EINTR A subroutine call was interrupted.

EINVAL An invalid argument was received.

EIO An unrecoverable I/O error occurred on the requested data transfer.

ENODEV An open was requested for an invalid channel.

ENOMEM The driver could not allocate memory for use in the data transfer.

ENXIO An operation was requested for an invalid minor device number.

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 operations 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 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 buffer to begin to write, use the **EMSEEK** ioctl operation.

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 scan code 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 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 through 7) correspond to bits 0 through 7 of the Interrupt Status register. Bits 8 through 15 of the mask correspond to bits 0 through 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 **/usr/include/sys/devinfo.h** file, to the user-specified area. The `devtype` field has a value of **DD_EM78**, which is defined in the **devinfo.h** file, and the flag field value has a value of 0.

EMSEEK Sets the offset into the communication adapter buffer to begin a **read** or **write** subroutine call.

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 not valid.
EINTR	A subroutine call was interrupted.
EINVAL	An invalid argument was specified on an **ioctl** call.
EL3RST	A **reset** command was received by the communications adapter.
ENOCONNECT	The connection to the control unit stopped 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 a minor device number that is not valid.

Implementation Specifics

This special file requires the IBM 3270 Connection Adapter.

The **3270cn** special file is part of Base Operating System (BOS) Runtime.

Related Information

Special Files Overview.

bus Special File

Purpose

Provides access to each of the hardware buses by way of the machine I/O device driver.

Description

The **bus** special files consist of a pseudo-driver in the kernel that allows a privileged user to access each hardware I/O bus. This is done indirectly by using the **ioctl** subroutine. The calling process, however, must have the appropriate system privilege to open the **bus** special files.

For additional information on **bus** special files, see device configuration documentation in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts* and machine device driver documentation in *AIX Version 4.3 Technical Reference: Kernel and Subsystems Volume 1*.

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.

There is at least one **bus** special file, usually the **/dev/pci0** or the **/dev/bus0** special file. This file accesses the primary hardware bus. One **bus** special file exists for each hardware bus on the machine. Each **bus** special file gains access to the corresponding hardware bus, and exists only if the hardware bus is present or was present at one time. Run the following command to generate a list of all the defined **bus** special files for a machine:

```
lsdev -C -c bus -F name | xargs -i echo
/dev/{}
```

Implementation Specifics

The **bus** special file is part of Base Operating System (BOS) Runtime.

Related Information

The **ioctl** subroutine.

Special Files Overview.

Device Configuration Subsystem Programming Introduction, Machine Device Driver,

cd Special File

Purpose

Provides access to the CD-ROM device driver.

Description

The CD-ROM special file provides block and character (raw) access to disks in the CD-ROM drives.

The **r** prefix on a special file name means the drive is accessed as a raw device rather than a block device. Performing raw I/O with a compact disk requires the performance of all data transfers in multiples of the compact-disk logical block length. Also, all **lseek** subroutines made to the raw CD-ROM device driver must set the file offset pointer to a value that is a multiple of the specified logical block size.

CD-ROM Device Drivers

Compact disks, used in CD-ROM device drivers, are read-only media that provide storage for large amounts of data. The special files **/dev/cd0**, **/dev/cd1**,... provide block access to compact disks. The special files **/dev/rcd0**, **/dev/rc1**,... provide character access.

When a CD-ROM disc is ejected from the drive for a mounted CD-ROM file system, the files on the compact disc can no longer be accessed. Before these files can be accessed again, the file systems mounted from the CD-ROM must be unmounted. Processes having files open on these file systems should be exited. Processes having current directories on these file systems should be moved. If these actions do not work, perform a forced unmount.

Another problem that results from ejecting the CD-ROM disc for a mounted CD-ROM file system is that InfoExplorer and the **man** command can become unresponsive. Reinserting the CD-ROM disc will not fix the problem. All InfoExplorer processes (graphical and ASCII) should be exited and the file system should be forced unmounted and mounted again. Afterwards, InfoExplorer and any **man** commands can be started again.

Device-Dependent Subroutines

Most CD-ROM operations are implemented using the **open**, **read**, and **close** subroutines. However, for some purposes, use of the **openx** (extended) subroutine is required.

openx Subroutine	The **openx** subroutine is supported to provide additional functions to the open sequence. The **openx** subroutine requires appropriate authority to start. Attempting to execute this subroutine without the proper authority results in a return value of -1, with the **errno** global variable set to **EPERM**.
ioctl Subroutine	The **IOCINFO** ioctl operation is defined for all device drivers that use the **ioctl** subroutine. The remaining ioctl operations are all physical volume device-specific operations. Diagnostic mode is not required for the following operation. The **IOCINFO** operation returns a **devinfo** structure, which is defined in the **devinfo.h** file.

Error Codes

In addition to the error codes listed for the **ioctl**, **open**, **read**, and **write** subroutines, the following error codes are also 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 was opened with a mode other than read-only.
EINVAL	An *nbyte* parameter to a **read** subroutine is not an even multiple of the block size.
EINVAL	A sense-data buffer length greater than 255 is not valid for a **CDIOCMD** ioctl operation.
EINVAL	A data buffer length greater than that allowed by the drive is not valid for a **CDIOCMD** ioctl operation.
EINVAL	An attempt was made to configure a device that is still open.
EINVAL	An illegal configuration command has been given.
EMFILE	An **open** call has been attempted for a SCSI adapter that already has the maximum permissible number of open devices.
ENOTREADY	There is no compact disk in the drive.
ENODEV	An attempt was made to access a device that is not defined.
ENODEV	An attempt was made to close a device that has not been defined.
EMEDIA	The media was 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 CD-ROM disk was 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 **cd** special file is part of Base Operating System (BOS) Runtime.

Related Information

The **close** subroutine, **ioctl** subroutine, **lseek** subroutine, **open** subroutine, **read** subroutine.

Special Files Overview.

The **scdisk** SCSI Device Driver in *AIX Version 4.3 Technical Reference: Kernel and Subsystems Volume 1*.

SCSI Subsystem

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 the appropriate permissions. 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 is typically a terminal or display located near the system unit. It has two functions in the operating system. First, it provides access to the system when it is operating in a non-multiuser mode. (This would be the case during maintenance and diagnostic sessions.) A console login is also normally provided on this device for all operating system run levels.

Second, the system console displays messages for system errors and other problems requiring intervention. These messages are generated by the operating system and its various subsystems when starting 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 Driver Configuration Support

Console driver configuration support allows the system console to be assigned or reassigned to a specified device or file. Such 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 startup.

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 that provides 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 started again.
- 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 started.

The **chcons** (change console) command can be used to switch the system console output to a different device or file for the next startup. This command does not affect the current console selection, but becomes effective when the system is started again.

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 startup. This action 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.

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, the opens are then passed 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.

An *ext* parameter passed using the **openx** subroutine is passed to the device driver supporting the console target or else ignored. (The latter is true if the console selection is a file.)

The **close** subroutine support is standard.

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.

An *ext* parameter passed to the **ioctlx** subroutine 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.)

read and write Subroutines

The **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 is unsuccessful, the console device driver switches back to the console device or file from which the system was started and tries the write again. This prevents important system messages from being lost in case the temporary console target is unavailable or unsuccessful. The console device driver should stay connected to the original system device until another **swcons** command is issued.

If the current console selection is a device, it redirects the **read** subroutine call. If the current console selection is a file, the **read** call is rejected with an error (**EACCES**).

An *ext* parameter passed to the **readx** or **writex** subroutine is passed to the device driver supporting the console target, or else ignored. (The latter is true if the console selection is a file.)

Console Output Logging

All output sent to the console is logged to a system log file. Only output sent to the console is logged. Any output sent to a device acting as the console is not logged. This means that system informational, error, and intervention-required messages are captured (logged), while other types of output seen at the console are not; e.g., getty output, smitty output, user interaction at the console device, etc.

The log file is based on the **alog** format; this format allows the file to wrap after it attains a predetermined maximum size. The **alog** command is typically used to view the console log file. The console log file deviates from the normal alog format in that each record of the file contains, in addition to the logged text, the user id who wrote to the console and the epoch time when it was written. The epoch time is formatted and displayed in the user's locale date and time when the file is output by the **alog** command.

When the console device is configured or when any modification is made to the console log file, ownership of the file is set to root and permissions are set to 622 to match that of the console device driver special file. The root user can modify the ownership or permissions, but they will not persist across boots.

The **swcons** command is used to make changes to console logging parameters during system operation; these changes are rescinded at the next console device configuration (typically reboot), and the original console logging parameters are reinstated.

The **chcons** command is used to make changes to the console logging parameters for the next console device configuration (typically reboot). These changes do not apply to the current running system.

The console logging facility can also be configured using the **alog** command. When the **alog -C** flag is used, changes are effective in the current running system and are persistent across boots. When the **-s** flag is used (without) the **-C**) to change the file size, the file is changed immediately but this change is not saved in the ODM and is not persistent across boots.

The parameters that control the console logging facility are the pathname of the log file, the maximum size of the log file, and the verbosity index for logging. Restrictions on these parameters are:

- the log file path must be absolute
- the maximum file size must not exceed the current free space of the file system on which it is stored (and the user entered value is rounded up to the nearest 4K boundary)
- verbosity values are 0-9 with any value greater than 0 indicating that all console output is to be recorded.

Console Output Tagging

A facility is provided to prefix each console output message with the effective user ID of the user that sent the message to the console. Only output sent to the console is tagged, any output sent to the device acting as the console is not.

Both the **swcons** command and the **chcons** commands can be used to enable and disable console output tagging with the same caveats about the persistence of the values applying as mentioned above in Console Output Logging.

The console output tagging verbosity value is limited to the range 0-9. Any value greater than 0 causes all console output to be tagged.

Implementation Specifics

The **console** special file is part of Base Operating System (BOS) Runtime.

Files

/dev/null	Provides access to the null device.

Related Information

The **chcons** command, **getty** command, **lscons** command, **swcons** command, **alog** command.

Special Files Overview.

The **consdef** file.

The **close** subroutine, **ioctl** subroutine, **lseek** subroutine, **open** subroutine, **poll** subroutine, **read** subroutine, **select** subroutine,

dials Special File

Purpose

Provides access to the dials.

Description

The **dials** special file is the application interface to the dials. It allows the applications to receive operator input from the dials and to set the granularity of the dials.

Configuration

Standard configuration methods are provided for the **dials** special file. The user cannot enter configurable attributes by way of the command line.

Usage Considerations

open

An **open** subroutine call specifying the **dials** special file is processed normally except that the *Oflag* and *Mode* parameters are ignored. An open request is rejected if the special file is already opened or if a kernel extension attempts to open the **dials** special file. All dials inputs are flushed following an open call until an input ring is established.

read and write

The **dials** special file does not support read or write subroutine calls. Input data is obtained from the dials via the input ring. The read and write subroutine calls behave the same as read or write to **/dev/null**. See "LFT Input Ring" in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts* for how to use the input ring.

ioctl

The dials special file supports the following **ioctl** operations:

IOCINFO	Returns the **devinfo** structure.
DIALREGRING	Registers input ring.
DIALRFLUSH	Flushes input ring.
DIALSETGRAND	Sets dial granularity.

Error Codes

The error codes can be found in the **/usr/include/sys/errno.h** file.

EFAULT	Indicates insufficient authority to access address or invalid address.
EIO	Indicates I/O error.
ENOMEM	Indicates insufficient memory for required paging operation.
ENOSPC	Indicates insufficient file system or paging space.
EINVAL	Indicates invalid argument specified.
EINTR	Indicates request interrupted by signal.
EPERM	Indicates a permanent error occurred.
EBUSY	Indicates device busy.
ENXIO	Indicates unsupported device number.
ENODEV	Indicates unsupported device or device type mismatch.

Implementation Specifics

The **dials** special file is part of Base Operating System (BOS) Runtime.

Files

/usr/include/sys/inputdd.h	Contains declarations for ioctl commands and input ring report format.

Related Information

The **GIO** special file, **kbd** special file, **lpfk** special file, **mouse** special file, **tablet** special file.

The **close** subroutine, **ioctl** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

Special Files Overview .

Graphic Input Devices Subsystem Overview in

dump Special File

Purpose

Supports system dump.

Syntax

#include <sys/dump.h>

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

Related Information

The **dmp_add** kernel service, **dmp_del** kernel service.

RAS Kernel Services in *AIX Kernel Extensions and Device Support Programming Concepts*.

entn Special File

Purpose

Provides access to Ethernet high-performance LAN adapters by way of the 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 LAN. The device handler supports up to four adapters, each of which may be running either or both of the standard Ethernet and 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 can 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** subroutines are discussed based on the mode in which the device handler is operating.

Subroutine Support

The Ethernet device handler supports the **open** and **close**, **read**, **write**, and **ioctl** subroutines in the following manner:

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**). However, there are no particular considerations for closing the special file. The special file name used in an **open** call depends upon how the device is to be opened. Types of special file names are:

/dev/ent*n*
: An **open** 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** 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.

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.

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.

ioctl Subroutine

The Ethernet device handler supports the following ioctl operations:

CCC_GET_VPD	Returns adapter vital product data (VPD) if available and valid.
CIO_GET_FASTWRT	Returns the parameters required to issue an **ent_fastwrt** call.
CIO_GET_STAT	Returns current adapter and device handler status.
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_START	Starts a session and registers a network ID.
ENT_SET_MULTI	Sets or clears a multicast address.
IOCINFO	Returns a device information structure to the user specified area. The `devtype` field value is **DD_NET_DH** and the `devsubtype` field is value **DD_EN**, as defined in the **sys/devinfo.h** file.

Error Codes

The following error codes may be returned when accessing the device handler through the **dev/ent*n*** special file:

EACCES	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	The address family is not supported by the protocol, or the multicast bit in the address is not set.
EAGAIN	The transmit queue is full.
EBUSY	The request is denied because the device is already opened in Diagnostic mode, or the maximum number of opens was reached.
EEXIST	The define device structure (DDS) already exists.
EFAULT	An address or parameter was specified that is not valid.
EINTR	A subroutine call was interrupted.
EINVAL	A range or operation code that is not valid was specified, or the device is not in Diagnostic mode.
EIO	An I/O error occurred.
ENOBUFS	No buffers are available.
ENOCONNECT	A connection was not established.
ENODEV	The device does not exist.
ENOENT	There is no DDS to delete.
ENOMEM	The device does not have enough memory.
ENOMSG	No message of desired type was available.
ENOSPC	No space is left on the device (the multicast table is full).
ENOTREADY	The device is not ready, a **CIO_START** operation was not issued, or the operation was issued but did not complete.
ENXIO	The device does not exist, or the maximum number of adapters was exceeded.
EUNATCH	The protocol driver is not attached.

Implementation Specifics

This file functions with the Ethernet device handler.

The **ent***n* special file is part of Base Operating System (BOS) Runtime.

Related Information

Error Logging Special Files

Purpose

Support error logging.

Description

The **error** and **errorctl** special files support the logging of error events. Minor device 0 (zero) of the **error** special file is the interface between processes that log error events and the **errdemon** (error daemon). Error records are written to the **error** special file by the **errlog** library routine and the **errsave and errlast** kernel services. 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 time stamp are recorded in nonvolatile random access memory (NVRAM). Therefore, in the event of a system crash, the last logged error 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.

The **ioctl** function interface for the **error** special file is provided by the **errorctl** special file. This interface supports stopping the error logging system, synchronizing the error logging system, and querying the status of the error special file.

Implementation Specifics

These files are part of the operating system.

Related Information

Special Files Overview in *AIX Version 4.3 Files Reference*

The **errclear** command, **errdead** command, **errdemon** command, **errinstall** command, **errlogger** command, **errmsg** command, **errpt** command, **errstop** command, **errupdate** command.

The **errlog** subroutine.

The **errsave and errlast** kernel services.

RAS Kernel Services in *AIX Kernel Extensions and Device Support Programming Concepts*.

Error Logging

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 usually specifies 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/fd0** and **/dev/fd1** 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 until it has read the diskette successfully or until it has tried all the possibilities supported for the drive type by the device driver.

An **r** prefix 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 **fd** special file supports three diskette drives: the 1.2MB, 5.25-inch diskette drive, and the 1.44MB and 2.88MB, 3.5-inch diskette drives. All **fd** special file names (except the generic special files **/dev/fd0**, **/dev/fd1**, **/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 the *Prefix*, *X*, and *Y* variables 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 followed by numeric characters is allowed. Possible values are:

- **h** Highest density supported by the drive type
- **l** Lowest density supported by the drive type
- **.9** 9 sectors per track (all three drive types)
- **.15** 15 sectors per track (1.2MB, 5.25-inch drive only)
- **.18** 18 sectors per track (both 3.5-inch drive types)
- **.36** 36 sectors per track (2.88MB, 3.5-inch drive only)

1.44MB, 3.5-inch Diskette Special Files

Ten different special files are available for use with the 1.44MB, 3.5-inch diskette drive. The default diskette type assumed for this drive type is a double-sided, 80-cylinder, 18 sectors-per-track diskette.

An **h** or **.18** as the suffix of the special file name (for example, `/dev/rfd0h` or `/dev/fd0.18`) forces a diskette to be treated as a double-sided, 80-cylinder, 18 sectors-per-track diskette. An **l** or **.9** as the suffix of the special file name (for example, `/dev/fd1l` or `/dev/rfd0.9`) forces a diskette to be treated as a double-sided, 80-cylinder, 9 sectors-per-track diskette.

2.88MB, 3.5-inch Diskette Special Files

Twelve different special files are available for use with the 2.88MB, 3.5-inch diskette drive. The default diskette type assumed for this drive type is a double-sided, 80-cylinder, 36 sectors-per-track diskette.

An **h** or **.36** as the suffix of the special file name (for example, `/dev/fd1h` or `/dev/fd0.36`) forces a diskette to be treated as a double-sided, 80-cylinder, 36 sectors-per-track diskette. An **l** or **.9** as the suffix of the special file name (for example, `/dev/rfd0l` or `/dev/fd1.9`) forces a diskette to be treated as a double-sided, 80-cylinder, 9 sectors-per-track diskette. A suffix of **.18** (for example, `/dev/fd1.18`) forces a diskette to be treated as a double-sided, 80-cylinder, 18-sectors-per-track diskette.

1.2MB, 5.25-inch Diskette Special Files

Ten different special files are available for use with the 1.2MB, 5.25-inch diskette drive. The default diskette type assumed for this drive type is a double-sided, 80-cylinder, 15 sectors-per-track diskette.

An **h** or **.15** as the suffix of the special file name (for example, `/dev/rfd1h` or `/dev/fd0.15`) forces a diskette to be treated as a double-sided, 80-cylinder, 15 sectors-per-track diskette. An **l** or **.9** as the suffix of the special file name (for example, `/dev/fd0l` or `/dev/rfd1.9`) forces a diskette to be treated as a double-sided, 80-cylinder, 9 sectors-per-track diskette.

Note: Regardless of the diskette drive type, an **h** as the suffix of the special file name forces a diskette to be treated as the highest capacity diskette supported by the drive type. When an **l** is used as the suffix of the special file name, the diskette is treated as the lowest capacity diskette supported by the drive type.

Usage Considerations

When using subroutines with the **fd** special file, consider the following items:

open and close subroutines	Only one process at a time can issue an **open** subroutine to gain access to a particular drive. However, all child processes created by a parent process that successfully opens a diskette drive inherit the open diskette drive.
read and write subroutines	No special considerations.

ioctl subroutines

The possible ioctl operations and their descriptions are:

IOCINFO	Returns a **devinfo** structure (defined in the **/usr/include/sys/devinfo.h** file) that describes the device.
FDIOCSINFO	Sets the characteristics of the device driver diskette to the values passed in the **fdinfo** structure, as defined in the **/usr/include/sys/fd.h** file.
FDIOCGINFO	Gets the device-driver diskette characteristics and returns the values in the **fdinfo** structure, as defined in the **/usr/include/sys/fd.h** file.
FDIOCFORMAT	Formats a diskette track. The diskette is formatted using data passed in an array of bytes. The length of this array is four 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. There are three ways to do this:

- 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 only when the **O_NDELAY** flag is 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 Base Operating System (BOS) Runtime.

Special Files Overview .

Related Information

The **close** subroutine, **ioctl** subroutine, **lseek** subroutine,

n

fddi Special File

Purpose

Provides access to the FDDI device driver by way of the FDDI device handler.

Description

The **fddi***n* special file provides access to the FDDI device handler that provides access to a FDDI local area network.

When accessing the FDDI device driver, the following information should be taken into account.

Driver Initialization and Termination

The device driver can be loaded and unloaded. The device driver supports the configuration calls to initialize and terminate itself.

Special File Support

Subroutine calls other than those made with the **open** and **close** subroutines are discussed based on the mode in which the device driver is operating.

Subroutine Support

The FDDI device driver provides specific support for the **open**, **close**, **read**, **write**, **ioctl**, **select**, and **poll** subroutines.

The device driver supports the **/dev/fddi***n* special file as a character-multiplex special file. The special file must be opened for both reading and writing. 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/fddi*n*	Starts the device driver for the selected port.
/dev/fddi*n*/**D**	Starts the device driver for the selected port in Diagnostic mode.
/dev/fddi*n*/**C**	Starts the device driver for the selected port in Diagnostic Configuration mode.

Error Codes

The following error conditions may be encountered when accessing the FDDI device driver through the **/dev/fddi***n* special file. The error codes can be found in the **/usr/include/sys/errno.h** file.

ENODEV Indicates that an invalid minor number was specified.

EINVAL Indicates that an invalid parameter was specified.

ENOMEM Indicates that the device driver was unable to allocate the required memory.

EINTR Indicates that a system call was interrupted.

EPERM Indicates that the Diagnostic mode open request was denied because the device was already open.

EACCES Indicates one of the following:

- A non-privileged user tried to open the device in Diagnostic mode.
- An illegal call from a kernel-mode user.
- An illegal call from a user-mode user.

ENETDOWN Indicates one of the following:

- The network is down. The device is unable to process the requested operation.
- An unrecoverable hardware error.

ENETUNREACH Indicates that the device is in Network Recovery mode and is unable to process the requested operation.

ENOCONNECT Indicates that the device has not been started.

EAGAIN Indicates that the transmit queue is full.

EFAULT Indicates that an invalid address was supplied.

EIO Indicates an error. See the status field for detailed information.

EMSGSIZE Indicates that the data was too large to fit into the receive buffer and that no *ext* parameter was supplied to provide an alternate means of reporting this error with a status of **CIO_BUF_OVFLW**.

Implementation Specifics

The FDDI device driver is a separately orderable feature code with the Base Operating System.

Related Information

The **close** subroutine, **ioctl** subroutine, **open** subroutine, **poll** subroutine, **read** subroutine, **select** subroutine, and **write** subroutine.

GIO Special File

Purpose

Provides access to the graphics I/O (GIO) adapter.

Description

The **GIO** special file is the application interface to the GIO adapter. The **GIO** special file provides applications with the ability to determine what I/O devices are attached to the GIO adapter.

Configuration

Standard configuration methods are provided for the **GIO** special file. User configurable attributes for the **GIO** special file do not exist.

Usage Considerations

The **open** subroutine call specifying the **GIO** special file is processed normally except that the *Oflag* and *Mode* parameters are ignored. An **open** request is rejected if the special file is already opened or if a kernel extension attempts to open the **GIO** special file.

Calls to the **read** and **write** routines behave as if the call was made to the **/dev/null** file.

The **GIO** special file supports the following functions with ioctls:

IOCINFO	Returns the **devinfo** structure.
GIOQUERYID	Returns the identifier of device connected to the GIO adapter.

Error Codes

The following error codes can be found in the **/usr/include/sys/errno.h** file:

EFAULT	Indicates insufficient authority to access address or invalid address.
EIO	Indicates an I/O error.
ENOMEM	Indicates insufficient memory for required paging operation.
ENOSPC	Indicates insufficient file system or paging space.
EINVAL	Indicates that an invalid argument was specified.
EINTR	Indicates a request interrupted by signal.
EPERM	Indicates a permanent error occurred.
EBUSY	Indicates the device is busy.
ENXIO	Indicates an unsupported device number.
ENODEV	Indicates an unsupported device or device type mismatch occurred.

Implementation Specifics

The **GIO** special file is part of Base Operating System (BOS) Runtime.

Files

/usr/include/sys/inputdd.h	Contains the ioctl commands.

Related Information

The **close** subroutine, **ioctl** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

The **dials** special file, **lpfk** special file.

Special Files Overview

ide Special File

Purpose

Provides access to the Integrated Device Electronics (IDE) adapter driver.

Description

The **ide** special file provides an interface to an attached IDE Bus. This special file should not be opened directly by application programs. The **/dev/ide0**, **/dev/ide1**, ... **/dev/ide***n* files are the **ide** special files.

Implementation Specifics

IDE Adapter Device Driver in *AIX Version 4.3 Technical Reference: Kernel and Subsystems Volume 2* provides the implementation specifics for the IDE adapter.

The **ide** special file is part of Base Operating System (BOS) Runtime.

Related Information

Special Files Overview.

Integrated Device Electronics (IDE) Subsystem Overview and Direct Access Storage Device Subsystem Overview in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts.*

kbd Special File

Purpose

Provides access to the natively attached keyboard.

Description

The **kbd** special file is the interface to the native keyboard. It provides an interface through which applications can receive operator input from the keyboard, control the keyboard LED's, and changing various keyboard parameters. The special file also allows an application to send an audible signal to the operator via the speaker located within the keyboard.

Configuration

The sound volume, click volume, typematic rate and typematic delay are configurable by the application through the **ioctl** subroutine. These changes are not reflected in the ODM database. To change these attributes in the ODM database, use the **chhwkbd** command.

Usage Considerations

open

This subroutine call creates a channel between the application and the natively attached keyboard. Two channels are supported. The open subroutine call is processed normally except that the *MODE* and *Oflag* parameters are ignored. All keyboard input is flushed until an input ring is established. Only the input ring associated with the most recent open receives input reports.

close

When the **kbd** device has been opened twice, input is reported through the input ring registered previous to the first **open**, after the **close** subroutine call.

read and write

The keyboard device driver does not return nor accept data via **read** and **write**. These calls behave as if the call was made to **/dev/null**. Input data is received from the input drivers via the input ring.

ioctl

The keyboard device driver supports the following ioctl commands:

IOCINFO	Return **devinfo** structure.
KSALARM	Sound alarm.
KSCFGCLICK	Control keyboard click.
KSDIAGMODE	Enable/disable diagnostics mode (user mode only).
KSLED	Set/reset keyboard LED’s.
KSKAP	Enable/disable keep alive poll (user mode only).
KSKAPACK	Acknowledge keep alive poll (user mode only).
KSQUERYID	Query keyboard device identifier.
KSQUERYSV	Query keyboard service vector (kernel mode only).
KSREGRING	Register input ring.
KSRFLUSH	Flush input ring.
KSTDELAY	Set typamatic delay.
KSTRATE	Set typamatic rate.
KSVOLUME	Set alarm volume

Error Codes

The error codes can be found in the **/usr/include/sys/errno.h** file.

EFAULT	indicates insufficient authority to access address or invalid address.
EIO	indicates that an I/O error occurred.
ENOMEM	indicates there was insufficient memory for required paging operation.
ENOSPC	indicates there was insufficient file system or paging space.
EINVAL	indicates that an invalid argument was specified.
EINTR	indicates the request was interrupted by signal.
EPERM	indicates that a permanent error occurred.
EBUSY	indicates the device is busy.
ENXIO	indicates unsupported device number was specified.
ENODEV	indicates an unsupported device or device type mismatch.

Implementation Specifics

The **kbd** special file is part of Base Operating System (BOS) Runtime.

Files

/usr/include/sys/inputdd.h	Contains declarations for ioctl commands and input ring report format.

Related Information

The **close** subroutine, **ioctl** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

Special Files Overview

lft Special File

Purpose

Provides character-based terminal support for the local graphics display and keyboard.

Description

The **lft** file is the application interface to the "Low Function Terminal (LFT) Subsystem". It provides support for a VT100-like terminal which is associated with the local graphics display and keyboard. It provides only character operations and is designed to be used during system installation, startup, shutdown, and stand-alone diagnostics.

The terminal supports a single logical screen size of 80 characters and 25 lines and a single color. Dynamic reconfiguration is not supported, configuration changes take effect at the next system startup. In the cases when multiple fonts may be used to achieve the 80x25 screen size, the user may set which font is used with the next system restart. See "LFT User Commands" for details of the available commands.

When multiple displays are available, the LFT Subsystem initially uses the default display. The user may change to another display and set the default display. See "LFT User Commands" for details of the available commands.

Usage Considerations

The LFT device driver supports the **lft** special file. The device driver is a streams based driver. It handles only the system attached keyboard and graphics displays.

Sharing Displays with Graphic Subsystem

Certain LFT **ioctl** commands allow graphics subsystems, such as AIXwindows, to obtain exclusive use of the displays, a right initially held by the LFT. However, this is done by the Rendering Context Manager (RCM) on behalf of the graphics subsystem. See "Rendering Context Manager" for details of the procedure for becoming a *graphics process*.

Subroutine Support

The **lft** special file supports the **open**, **close**, **read**, **write**, and **ioctl** subroutines.

ioctl system call

The functions performed by the **ioctl** commands fall into three categories:

- Sharing devices between the lft and a graphic subsystem such as AIXwindows
- Query information about configured displays and keyboard devices
- Compatibility with the common tty ioctl commands

Sharing devices

IOCINFO	The IOCINFO ioctl operation is defined for all device drivers that use the ioctl subroutine. The IOCINFO operation returns a **devinfo** structure, which is defined in the **devinfo.h** file.
LFT_SET_DEFLT_DISP	Sets the default display.
LFT_ACQ_DISP	Acquire display for exclusive use.
LFT_REL_DISP	Release display.
LFT_DIAG_OWNER	Acquire display for diagnostics.

Query information about configured displays and keyboard devices

LFT_QUERY_LFT	Query common LFT information.
LFT_QUERY_DISP	Query display information.

Compatibility with the common tty ioctl commands

TCSAK

TCGETA

TCSETAW

TCSETAF

TCSETA

TIOCGWINSZ

TIOCSWINSZ

TXTTYNAME

TSCBRK

Implementation Specifics

The **lft** special file is part of Base Operating System (BOS) Runtime.

Related Information

Low Function Terminal (LFT) Subsystem Overview in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts*.

rcm and **kbd** Special Files.

Special Files Overview

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, the application must 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, each form-feed character is replaced with a line-feed character, 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 character. 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, lowercase characters are converted to uppercase. 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 ... (ellipsis). 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 be turned off in the following two cases:

- An application needs 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.

IGNOREPE If IGNOREPE mode is on, the device driver allows writes to the device regardless of the state of the PE (paper-end) line on the parallel interface. An application can make use of this mode, for example, to change the paper tray of a printer under software control when detecting that the printer is out of paper.

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. RPTERR 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** error code.

If printing is canceled and the printer is in PLOT mode, it is the application must send the final form-feed character to eject the partial page. If the printer is not in PLOT mode, the final form-feed character 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 code. This return allows the application to get the printer status and possibly display an error message.

Note: When a device driver returns an incomplete operation with an error code (as previously described), the application must resend 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.

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 child processes created by a parent 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.

read and write Subroutines

The **read** subroutine is not implemented for the native I/O parallel port.

When printing to a parallel printer that is offline, the **write** subroutine may return one fewer than the actual number of bytes that are buffered and ready to be written when the printer is put back online. This is used as a mechanism to indicate to the calling application that there is a problem with the printer requiring user intervention, possibly allowing the user to put the printer online and continue with printing. In this situation, no error is returned by the **write** subroutine.

ioctl Subroutine

The possible ioctl operations and their descriptions are:

IOCINFO Returns a structure defined in the **/usr/include/sys/devinfo.h** file, which describes the device.

LPQUERY Provides access to the printer status. Refer to the **/usr/include/sys/lpio.h** 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.

LPRGET Returns the page length, width and indentation values. These values are used by the device driver when PLOT mode is not set. The default printer backend, **piobe**, sends all print jobs with PLOT mode set. The **LPRGET** operation uses the **lprio** structure, as defined in the **/usr/include/sys/lpio.h** 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** file.

Note: This operation is supported for compatibility reasons. The use of the **tcgetattr** subroutine is recommended.

LPRGTOV Gets the current time-out value and stores it in the **lptimer** structure defined in the **/usr/include/sys/lpio.h** file. The time-out value is measured in seconds.

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 **/usr/include/sys/lpio.h** 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 **/usr/include/sys/lpio.h** file.

LPRSET Sets the page length, width and indent values. These values are used by the device driver when PLOT mode is not set. The default printer backend, **piobe**, sends all print jobs with PLOT mode set. The **LPRSET** operation uses the **lprio** structure, as defined in the **/usr/include/sys/lpio.h** 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.

Note: This operation is supported for compatibility reasons. The use of the **tcsetattr** subroutine is recommended.

LPRSTOV Sets the time-out value. The *arg* parameter to this ioctl operation points to a **lptimer** structure defined in the **/usr/include/sys/lpio.h** file. The time-out value must be given in seconds.

Implementation Specifics

The Printer Addition Management Subsystem: Programming Overview in *AIX Kernel Extensions and Device Support Programming Concepts* provides more information on implementation specifics.

The **lp** special file is part of Base Operating System (BOS) Runtime.

Special Files Overview .

Related Information

The **lsattr** command, **piobe** command, **splp** command.

lpfk Special File

Purpose

Provides access to the lighted program function key (LPFK) array.

Description

The **lpfk** special file is the application interface to the lighted program function keys. It allows the application to receive operator input from the LPFKs and to illuminate and darken each key in the array.

Configuration

Standard configuration methods are provided for the **lpfk** special file. The user cannot enter configurable attributes by way of the command line.

Usage Considerations

open

An **open** subroutine call specifying the **lpfk** special file is processed normally except that the *Oflag* and *Mode* parameters are ignored. An open request is rejected if the special file is already opened or if a kernel extension attempts to open the **lpfk** special file. All LPFK inputs are flushed following an open call until an input ring is established.

read and write

The **lpfk** special file does not support **read** or **write** subroutine calls. Instead, input data is obtained from the LPFKs through the input ring. The **read** and **write** subroutine calls behave the same as **read** and **write** functions of the **/dev/null** file. See "LFT Input Ring" in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts* for how to use the input ring.

ioctl

The **lpfk** special file supports the following **ioctl** operations:

IOCINFO	Returns the **devinfo** structure.
LPFKREGRING	Registers input ring.
LPFKRFLUSH	Flushes input ring.
LPFKLIGHT	Sets key lights.

Error Codes

The error codes can be found in the **/usr/include/sys/errno.h** file.

EFAULT	Indicates insufficient authority to access address, or invalid address.
EIO	Indicates I/O error.
ENOMEM	Indicates insufficient memory for required paging operation.
ENOSPC	Indicates insufficient file system or paging space.
EINVAL	Indicates invalid argument specified.
EINTR	Indicates request interrupted by signal.
EPERM	Indicates a permanent error occurred.
EBUSY	Indicates device busy.
ENXIO	Indicates unsupported device number.
ENODEV	Indicates unsupported device, or device type mismatch.

Implementation Specifics

The **lpfk** special file is part of Base Operating System (BOS) Runtime.

Files

/usr/include/sys/inputdd.h	Contains declarations for ioctl commands and input ring report format

Related Information

The **dials** special file, **GIO** special file, **kbd** special file, **mouse** special file, and **tablet** special file.

The **close** subroutine, **ioctl** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

Special Files Overview

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 with each logical volume in a volume group.

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.

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 (128KB) 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

> **Attention:** Data corruption, loss of data, or loss of system integrity (system crashes) 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 the operating system and are solely for system use in managing file systems, paging devices and logical volumes. They should not be used for other purposes. Additional information concerning the use of special files may be obtained in "Understanding I/O Access through Special Files" in *AIX Kernel Extensions and Device Support Programming Concepts*.

open and close Subroutines

No special considerations.

Extension Word Specification for the readx and writex Subroutines

The *ext* parameter for the **readx** and **writex** extended I/O subroutines indicates specific physical or logical operations, or both. The upper 4 bits of the *ext* parameter are reserved for internal LVDD use. The value of the *ext* parameter is defined by logically ORing values from the following list, as defined in the **/usr/include/sys/lvdd.h** file:

WRITEV	Perform physical write verification on this request. This operation can be used only with the **writex** subroutine.
RORELOC	For this request, perform relocation on existing relocated defects only. Newly detected defects should not 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 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. When synchronizing a striped logical volume, the data returned is not useable by the application because the logical track group is not read on a striped basis.
LV_READ_BACKUP	Read only the mirror copy that is designated as the backup mirror copy.
LV_WRITE_BACKUP	Write only the mirror copy that is designated as the backup mirror copy.
LV_READ_ONLY_C1	Read only copy one of the data.
LV_READ_ONLY_C2	Read only copy two of the data.
LV_READ_ONLY_C3	Read only copy three of the data.
LV_READ_STALE_C1	Read only copy one of the data even if it is stale.
LV_READ_STALE_C2	Read only copy two of the data even if it is stale.
LV_READ_STALE_C3	Read only copy three of the data even if it is stale.

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 128KB (the 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 an LP.

Normal I/O operations do not need to supply the *ext* parameter and can use the **read** and **write** subroutines.

IOCINFO ioctl Operation

The **IOCINFO** ioctl operation returns the **devinfo** structure, as defined in the **/usr/include/sys/devinfo.h** 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** 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

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 **/usr/include/sys/lvdd.h** 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

PBUFCNT ioctl Operation

The **PBUFCNT** ioctl operation increases the size of the physical buffer header, **pbuf**, pool that is used by LVM for logical-to-physical request translation. The size of this pool is determined by the number of active disks in the system, although the pool is shared for request to all disks.

The **PBUFCNT** ioctl operation can be issued to any active volume group special file, for example **/dev/***VolGrpName*. The parameter passed to this ioctl is a pointer to an unsigned integer that contains the *pbufs-per-disk* value. The valid range is 16 - 128. The default value is 16. This value can only be increased and is reset to the default at IPL. The size of the **pbuf** pool is not reduced when the number of active disks in the system is decreased.

The **PBUFCNT** ioctl operation returns the following:

EINVAL	Indicates an invalid parameter value. The value is larger than the maximum allowed, or smaller than or equal to the current value.
EFAULT	Indicates that the copy in of the parameter failed.
LVDD_ERROR	An error occurred in allocating space for additional buffer headers.
LVDD_SUCCESS	Indicates a successful ioctl operation.

LV_INFO ioctl Operation

The LV_INFO ioctl operation returns information about the logical volume in question. This ioctl operation only applies to AIX Versions 4.2.1 and later.

The caller supplies the logical volume special file in the system open call and the information is returned via the **lv_info** structure, as defined in the **/usr/include/sys/lvdd.h** file. This structure contains the following fields:

vg_id	Volume group ID of which the logical volume is a member
major_num	Major number of logical volume
minor_num	Minor number of the logical volume
max_lps	Maximum number of logical partitions allowed for this logical volume
current_lps	Current size of the lofical volume in terms of logical partitions
mirror_policy	Specifies the type of mirroring, if the logical volume is mirrored. Valid values are parallel, sequential, striped, and striped_parallel.
permissions	Specifies whether the logical volume is read only or read-write
bb_relocation	Specifies whether bad block relocation is activated for the logical volume
write_verify	Specifies whether the write verify command for writes to the logical volume is enforced
num_blocks	Number of 512 byte blocks that make up the logical volume. This value does not include mirrored logical volumes
mwcc	Specifies whether the mirrored write consistency check is enforced
mirr_able	Specifies whether the logical volume is capable of being mirrored
num_mirrors	Number of mirror copies for this logical volume
striping_width	Number of drives across which this logical volume is striped
stripe_exp	Stripe block exponent value
BACKUP_MIRROR	Backup mirror mask will be zero indicating there is not a backup copy active.
AVOID_C1	For the first copy,
AVOID_C2	For the second copy, or
AVOID_C3	For the third copy.
LV_READONLY_MIRROR	
LV_QRYBKPCOPY	Query for designated backup mirror copy.
LV_SETBKPCOPY	Designate backup mirror copy.
LV_FSETBKPCOPY	Force new designation for backup mirror copy. Used when there are stale partitions on either the active mirror or backup mirror.

The LV_INFO ioctl operation returns the following:

EFAULT Indicates that the copy of the parameter failed.

Error Codes

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 only 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 only to the **XLATE** ioctl operation.)

ENXIO No physical partition has been allocated to this copy (mirror). (This error type is relevant only to the **XLATE** ioctl operation.)

Implementation Specifics

The **lvdd** special file is part of Base Operating System (BOS) Runtime.

Related Information

The **close** subroutine, **ioctl** subroutine, **lseek** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

Special Files Overview .

mem or kmem Special File

Purpose

Provides privileged virtual memory read and write access.

Description

> **Attention:** When incorrect access to virtual memory is made through these files, process termination, a system crash, or loss of system data integrity can result.

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.

> **Note:** Programs accessing these special files must have appropriate privilege. Commercial application programs should avoid using the **/dev/mem** and **/dev/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 systems.

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.

> **Note:** On the Power PC 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

Attention: Use of this special file by application programs should be strictly avoided, as it is provided for diagnostic and problem determination procedures only.

The **mem** special file access is specific to the system on which AIX is executing.

Please refer to the Implementation Specifics section for details on the function provided by this special file.

Implementation Specifics

The **kmem** special file is part of Base Operating System (BOS) Runtime.

Process Address Space Regions for the /dev/kmem Special File

The Process Address Space Map illustrates the layout of process address space regions as accessed through the **/dev/kmem** special file on this 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 this operating system this function is indirectly provided by using the *ext* (extension) parameter on the **readx** and **writex** subroutine calls. When a **readx** or **writex** subroutine call associated with the **/dev/mem** special file is issued, the *ext* parameter must contain a valid segment register value as defined in the *POWERstation and POWERserver Hardware Technical Reference - General Information* documentation for the platform types(s) on which the program will be run. This allows the program to access all physical memory mapped by the page table as well as the platform-specific I/O (T=1) segments.

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 the **readx** or **writex** subroutine calls are issued, in order to specify the address offset.

If a **read** or **write** subroutine call is used with this special file, the access to memory is identical to that provided by the **/dev/kmem** special file.

The **mem** special file is part of Base Operating System (BOS) Runtime.

Files

/dev/mem	Provides privileged virtual memory read and write access.
/dev/kmem	Provides privileged virtual memory read and write access.

Related Information

The **crash** command, **iostat** command, **sar** command, **vmstat** command.

The **close** subroutine, **ioctl** subroutine, **knlist** subroutine, **lseek** subroutine, **open** subroutine, **poll** subroutine, **read** subroutine, **select** subroutine, **write** subroutine.

Special Files Overview

mouse Special File

Purpose

Provides access to the natively attached mouse.

Description

The **mouse** special file serves as an interface between the application interface and the system mouse. This special file provides the application with the ability to receive input from the mouse and allows the application to change mouse configuration parameters, such as mouse sampling rates and resolution.

Configuration

Standard configuration methods work with the mouse special file. No user configurable attribute commands exist for this special file. Applications that open the special file can modify device attribute dynamically using the appropriate ioctl interface; however, modifications are not saved in the configuration database.

Usage Considerations

The **open** subroutine call specifying the **mouse** special file is processed normally except that the *Oflag* and *Mode* parameters are ignored. The **open** request is rejected when the special file is already opened or when a kernel extension attempts to open the special file. All mouse inputs are flushed following an **open** subroutine call until an input ring is established. The mouse device is reset to the default configuration when an open request is made.

The **mouse** special file does not support the **read** or **write** subroutine calls. Instead, input data is obtained from the mouse via the input ring. The **read** and **write** subroutine calls behave the same as **read** or **write** to the **/dev/null** file.

The **mouse** special file supports the following functions with ioctls:

IOCINFO	Returns a **devinfo** structure.
MQUERYID	Returns the query mouse device identifier.
MREGRING	Specifies the address of the input ring and the value to be used as the source identifier when enqueueing reports on the ring.
MRFLUSH	Flushes the input ring.
MTHRESHOLD	Sets the mouse reporting threshold.
MRESOLUTION	Sets the mouse resolution.
MSCALE	Sets the mouse scale factor.
MSAMPLERATE	Sets the mouse sample rate.

Error Codes

The following error codes can be found in the **/usr/include/sys/errno.h** file:

EFAULT	Indicates insufficient authority to access an address or invalid address.
EIO	Indicates and I/O error.
ENOMEM	Indicates insufficient memory for required paging operation.
ENOSPC	Indicates insufficient file system or paging space.
EINVAL	Indicates invalid argument specified.
EINTR	Indicates that the request has been interrupted by a signal.
EPERM	Specifies a permanent error occurred.
EBUSY	Indicates a device is busy.
ENXIO	Indicates an unsupported device number.
ENODEV	Indicates an unsupported device or device type mismatch.
EACCES	Indicates that an open is not allowed.

Implementation Specifics

The **mouse** special file is part of Base Operating System (BOS) Runtime.

Files

/usr/include/sys/inputdd.h	Contains the ioctl commands.

Related Information

The **close** subroutine, **ioctl** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

Special Files Overview

mpcn Special File

Purpose

Provides access to the HDLC network device driver by way of the SDLC COMIO device driver emulator. This special file only applies to AIX Versions 4.2.1 and later.

Description

The **/dev/mpc**n character special file provides access to the HDLC network device driver via the SDLC COMIO device driver emulator in order to provide access to a synchronous network. The SDLC COMIO emulator device handler supports multiple HDLC network devices.

Usage Considerations

When accessing the SDLC COMIO emulator device handler, consider the following information.

Driver Initialization and Termination

The device handler can be loaded and unloaded. The handler supports the configuration calls to initialize and terminate itself.

Special File Support

The SDLC COMIO emulator device handler uses the **t_start_dev** and **t_chg_parms** structures defined in the **/usr/include/sys/mpqp.h** file to preserve compatibility with the existing GDLC, MPQP API, and SNA Services interface. However, only a subset of the *#define* values are supported for the following **t_start_dev** structure fields:

`phys_link`
: Indicates the physical link protocol. Only one type of physical link is valid at a time. The SDLC COMIO emulator device handler supports PL_232D (EIA-232D), PL_422A (EIA-422A/v.36), PL_V35 (V.35), PL_X21 (X.21 leased only), and PL_V25 (V.25bis EIA-422A autodial).

`data_proto`
: Identifies the data protocol. The SDLC COMIO emulator device handler supports only the SDLC DATA_PRO_SDLC_HDX (half duplex) and the DATA_PRO_SDLC_FDX (full duplex) values.

`baud_rate`
: Specifies the baud rate for transmit and receive clocks. The SDLC COMIO emulator device handler supports only external clocking where the DCE supplies the clock, and this field should be set to zero.

Subroutine Support

The SDLC COMIO emulator device handler supports the **open**, **close**, **read**, **write**, and **ioctl** subroutines in the following manner:

open and close Subroutines

The device handler supports the **/dev/mpc**n special file as a character-multiplex special file. The special file must be opened for both reading and writing (**O_RDWR**). No special considerations exist for closing the special file.

read Subroutine

Can take the form of a **read**, **readx**, **readv**, or **readvx** subroutine call. For this call, the device handler copies the user data in to the buffer specified by the caller.

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 wide area network using the HDLC network device driver.

ioctl Subroutine

The **ioctl** subroutine supports the following flags:

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 reliability, availability, and serviceability (RAS) counter values. These values are defined in the **/usr/include/sys/comio.h** file.
MP_CHG_PARMS	Permits the data link control (DLC) to change certain profile parameters after the SDLC COMIO device driver emulator is started.

Error Codes

The following error codes can be returned when gaining access to the device handler through the **/dev/mpc**n special file:

ECHRNG Indicates that the channel number is out of range.

EAGAIN Indicates that the device handler cannot transmit data because of a lack of system resources, or, because an error returned from the HDLC network device driver's transmit routine.

EBUSY Indicates that the device handler is already in use (opened/started) by another user.

EIO Indicates that the handler cannot queue the request to the adapter.

EFAULT Indicates that the cross-memory copy service failed.

EINTR Indicates that a signal has interrupted the sleep.

EINVAL Indicates one of the following:

- The port is not set up properly.
- The handler cannot set up structures for **write**.
- The port is not valid.
- A kernel process called a **select** operation.
- The specified physical-link parameter is not valid for that port.
- A kernel process called a **read** operation.

ENOMEM Indicates one of the following:

- No mbuf or mbuf clusters are available.
- The total data length is more than one 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 the 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 was not started successfully.
- The channel number is illegal.
- The driver control block pointer is null or does not exist.

Implementation Specifics

This file functions with the SDLC COMIO emulator device handler over the HDLC network device driver. It emulates the SDLC API (full and half duplex) of the Multiprotocol Quad Port (MPQP) device handler.

Related Information

The **close** subroutine, **open** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine.

Special Files Overview

2-Port Multiprotocol HDLC Network Device Driver Overview in *AIX Version 4.3 System Management Guide: Communications and Networks*

MPQP Device Handler

mpqi Special File

Purpose

Provides access to the Multiport Model 2 Adapter (MM2) device driver via SNA Services, GDLC, or user-written applications compatible with current MPQP Applications Programming Interface (API).

Description

The Multiport Model 2 devic e driver provides access to the **mpq**i special file through SNA Services, Generic Data Link Control, or through user-written applications.

Usage Considerations

When accessing the Multiport Model 2 device driver via these methods, consider the following information:

Driver Initialization and Termination

The device driver can be loaded and unloaded in the kernel in the same way as other communications device drivers. The device driver supports the configuration calls to initialize and terminate itself. Therefore, you must ensure that the device driver is initialized before using it. A listing of the device driver, either via SMIT or by using the **lsdev** command, should indicate the device driver state as *Available*.

Special File Support

The Multiport Model 2 device driver is a character I/O device and provides a special file entry in the **/dev** directory for file system access. The Multiport Model 2 device driver uses the **t_start_dev** and **t_chg_parms** structures defined in the **/usr/include/sys/mpqp.h** file to preserve compatibility with the existing GDLC, MPQP API and SNA Services interface. However, only a subset of the *#define* values is supported for the following **t_start_dev** structure fields:

`data_proto`	Identifies the data protocol. T he Multiport Model 2 device driver supports the SDLC DATA_PRO_SDLC_HDX value (indicating half duplex only) and the bisync DATA_PRO_BSC value. Bisync support is offered only in AIX Versions 4.2 and later.
`baud_rate`	Specifies the baud rate for transmit and receive clock. The Multiport Model 2 device driver only supports external clocking where the modem supplies the clock, and this field should be set to zero. However, when using SNA Services, this field is ignored when external clocking is specified in the physical link profile and does not need to be zero.

Related Information

Special Files Overview

MPQP Device Handler Interface Overview in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts*

Data Link

mpqn Special File

Purpose

Provides access to multiprotocol adapters by way of the Multiprotocol Quad Port (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 in relation to 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. Which special file name is used in an **open** call depends on 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 in Diagnostic mode for the selected port.

- 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 **/usr/include/sys/comio.h** file.
CIO_GET_STAT	Returns the current adapter and device handler status.
MP_START_AR	Puts the MPQP port into Autoresponse mode.
MP_STOP_AR	Permits the MPQP port to exit Autoresponse 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 Codes

The following error codes 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 direct memory accesses (DMAs) was reached, so that 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 is not opened or started.
- The port state is not opened for start of an ioctl operation.
- The port is not started or is not in data-transfer state.

EIO Indicates that the handler could not queue the request to the 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 not valid.
- A select operation was called by a kernel process.
- The specified physical-link parameter is not valid 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 adapter control block (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.

Related Information

The **close** subroutine, **open** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine, **ioctl** subroutine.

MPQP Device Handler Interface Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

Special Files Overview .

null Special File

Purpose

Provides access to the null device, typically for writing to the bit bucket.

Description

The **/dev/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

When using subroutines with the null device file, consider the following items:

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.
read and write subroutines	Data written to this file is discarded. Reading from this file always returns 0 bytes.
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 Base Operating System (BOS) Runtime.

Related Information

Special Files Overview .

The **close** subroutine, **ioctl** subroutine,

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 **/dev/nvram** character special file provides access to the machine device driver for accessing or modifying machine-specific nonvolatile RAM. The appropriate privilege is required to open the **nvram** special file. The **nvram** 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 **nvram** special file supports open, close, read, and ioctl operations.

> **Attention:** Application programs should not access the nonvolatile RAM. Since 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 startup and configuration information. Such a loss could require system administrative or maintenance task work to rebuild or recover.

For additional information concerning the use of this special file to access machine-specific nonvolatile RAM, see the "Machine Device Driver" in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

Usage Considerations

When using subroutines with the **nvram** special file, consider the following items.

open and close Subroutines

The machine device driver supports the **nvram** special file as a multiplexed character special file.

A special channel name of **base** can be used to read the base customize information stored as part of the boot record. The **nvram** special file must be opened with a channel name of base, as follows:

```
/dev/nvram/base
```

The special file **/dev/nvram/base** can only be opened once. When it is closed for the first time after a boot, the buffer containing the base customize information is free. Subsequent opens return a **ENOENT** error code.

read, write, and lseek Subroutines

The **read** subroutine is supported after a successful open of the **nvram** special file with a channel name of **base**. The **read** operation starts transferring data at the location associated with the base customization information and 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 can be used to change the starting read offset within the data area associated with the base customization information. The **write** subroutine is not supported on this channel and results in an error return of **ENODEV**.

ioctl Subroutine

ioctl commands can be issued to the machine device driver after the **/dev/nvram** special file has been successfully opened. The **IOCINFO** parameter 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 **/usr/include/sys/devinfo.h** file. The device type for this device driver is **DD_PSEU**.

Error Codes

The following error conditions can be returned when accessing the machine device driver using the **nvram** special file name:

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.
ENOMEM	A request was made with a user-supplied buffer that is too small for the requested data.

Security

Programs attempting to open the **nvram** special file require the appropriate privilege.

Implementation Specifics

For additional information concerning the data areas accessed by other channels associated with the **/dev/nvram** special file, see the "Machine Device Driver" in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

The **nvram** special file is part of Base Operating System (BOS) Runtime.

Files

/dev/nvram/base	Allows read access to the base customize information stored as part of the boot record.

Related Information

Special Files Overview .

The Device Configuration Subsystem Programming Introduction in *AIX Kernel Extensions and Device Support Programming Concepts*.

The **close** subroutine, **ioctl** subroutine, **lseek** subroutine,

omd Special File

Purpose

Provides access to the read/write optical device driver.

Description

The **omd** special file provides block and character (raw) access to disks in the read/write optical drive.

The **r** prefix 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 an optical disk requires that all data transfers be in multiples of the optical-disk logical block length. Also, all **lseek** subroutines that are made to the raw read/write optical device driver must set the file offset pointer to a value that is a multiple of the specified logical block size.

Read/Write Optical Device Driver

Read/write optical disks, used in read/write optical drives, are media that provide storage for large amounts of data. Block access to optical disks is achieved through the special files **/dev/omd0**, **/dev/omd1**, ... **/dev/omd***n*. Character access is provided through the special files **/dev/romd0**, **/dev/romd1**, ... **/dev/romd***n*.

When a read/write optical disk is ejected from the drive for a mounted read/write optical file system, the files on the optical disk can no longer be accessed. Before attempting to access these files again, perform the following steps for a file system mounted from the read/write optical disk:

1. Stop processes that have files open on the file system.
2. Move processes that have current directories on the file system.
3. Unmount the file system.
4. Remount the file system after reinserting the media.

If these actions do not work, perform a forced unmount of the file system; then, remount the file system.

> **Note:** Reinserting the read/write optical disk will not fix the problem. Stop all InfoExplorer processes (graphical and ASCII), and then forcibly unmount the file system. Then remount the file system. After performing this procedure, you can restart InfoExplorer and any **man** commands.

Device-Dependent Subroutines

Most read/write optical operations are implemented using the **open**, **read**, **write**, and **close** subroutines. However, for some purposes, use of the **openx** (extended) subroutine is required.

The openx Subroutine

The **openx** subroutine is supported to provide additional functions to the **open** sequence. Appropriate authority is required for execution. If an attempt is made to run the **openx** subroutine without the proper authority, the subroutine returns a value of **-**1 and sets the **errno** global variable to a value of **EPERM**.

The ioctl Subroutine

The **ioctl** subroutine **IOCINFO** operation returns the **devinfo** structure defined in the **/usr/include/sys/devinfo.h** file. The **IOCINFO** operation is the only operation defined for all device drivers that use the **ioctl** subroutine. Other **ioctl** operations are specific for the type of device driver. Diagnostic mode is not required for the **IOCINFO** operation.

Error Conditions

Possible **errno** values for **ioctl**, **open**, **read**, and **write** subroutines when using the **omd** special file include:

EACCES Indicates one of the following circumstances:

- An attempt was made to open a device currently open in Diagnostic or Exclusive Access mode.
- An attempt was made to open a Diagnostic mode session on a device already open.
- The user attempted a subroutine other than an **ioctl** or **close** subroutine while in Diagnostic mode.
- A **DKIOCMD** operation was attempted on a device not in Diagnostic mode.
- A **DKFORMAT** operation was attempted on a device not in Exclusive Access mode.

EBUSY Indicates one of the following circumstances:

- The target device is reserved by another initiator.
- An attempt was made to open a session in Exclusive Access mode on a device already opened.

EFAULT Indicates an illegal user address.

EFORMAT Indicates the target device has unformatted media or media in an incompatible format.

EINVAL Indicates one of the following circumstances:

- The **read** or **write** subroutine supplied an *nbyte* parameter that is not an even multiple of the block size.
- A sense data buffer length of greater than 255 bytes is not valid for a **DKIOWRSE** or **DKIORDSE ioctl** subroutine operation.
- The data buffer length exceeded the maximum defined in the **devinfo** structure for a **DKIORDSE**, **DKIOWRSE**, or **DKIOCMD ioctl** subroutine operation.
- An unsupported **ioctl** subroutine operation was attempted.
- An attempt was made to configure a device that is still open.
- An illegal configuration command has been given.
- A **DKPMR** (Prevent Media Removal), **DKAMR** (Allow Media Removal), or **DKEJECT** (Eject Media) command was sent to a device that does not support removable media.
- A **DKEJECT** (Eject Media) command was sent to a device that currently has its media locked in the drive.

EIO Indicates one of the following circumstances:

- The target device cannot be located or is not responding.
- The target device has indicated an unrecovered hardware error.

EMEDIA Indicates one of the following circumstances:

- The target device has indicated an unrecovered media error.
- The media was changed.

EMFILE Indicates an **open** operation was attempted for an adapter that already has the maximum permissible number of opened devices.

ENODEV Indicates one of the following circumstances:

- An attempt was made to access an undefined device.
- An attempt was made to close an undefined device.

ENOTREADY Indicates no read/write optical disk is in the drive.

ENXIO Indicates one of the following circumstances:

- The **ioctl** subroutine supplied an invalid parameter.
- A **read** or **write** operation was attempted beyond the end of the physical volume.

EPERM Indicates the attempted subroutine requires appropriate authority.

ESTALE Indicates a read-only optical disk was ejected (without first being closed by the user) and then either reinserted or replaced with a second disk.

ETIMEDOUT Indicates an I/O operation has exceeded the given timer value.

EWRPROTECT	Indicates one of the following circumstances: • An **open** operation requesting **read/write** mode was attempted on read-only media. • A **write** operation was attempted to read-only media.

Implementation Specifics

The scdisk SCSI Device Driver provides more information about implementation specifics.

The **omd** special file is part of Base Operating System (BOS) Runtime.

Files

/dev/romd0, **/dev/romd1**,**...**, **/dev/romd***n*	Provides character access to the read/write optical device driver.
/dev/omd0, **/dev/omd1**,**...**, **/dev/omd***n*	Provides block access to the read/write optical device driver.

Related Information

Special Files Overview.

scdisk SCSI Device Driver.

The **close** subroutine, **ioctl** subroutine, **lseek** subroutine,

opn Special File

Purpose

Provides a diagnostic interface to the serial optical ports by way of the Serial Optical Link device driver.

Description

The **op***n* character special file provides strictly diagnostic access to a specific serial optical port. The normal interface to the serial optical link is through the **ops0** special file.

Related Information

Serial Optical Link Device Handler Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

Special Files Overview .

ops0 Special File

Purpose

Provides access to the serial optical link by way of the Serial Optical Link Device Handler Overview.

Description

The Serial Optical Link device driver is a component of the Communication I/O subsystem. The device driver can support from one to four serial optical ports. An optical port consists of two separate pieces. The Serial Link Adapter is on the system planar, and is packaged with two to four adapters in a single chip. The Serial Optical Channel Converter plugs into a slot on the system planar and provides two separate optical ports.

The **ops0** special file provides access to the optical port subsystem. An application that opens this special file has access to all the ports, but does not need to be aware of the number of ports available. Each write operation will include a destination processor ID, and the device driver will route the data through the correct port to reach that processor. If there is more than one path to the destination, the device driver will use any link that is available, in case of a link failure.

Usage Considerations

When accessing the Serial Optical Link device driver, the following should be taken into account:

driver initialization and termination	The device driver may be loaded and unloaded. The device driver 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 driver is operating.

Subroutine Support

The Serial Optical Link device driver provides specific support for the **open**, **close**, **read**, **write**, and **ioctl** subroutines**.**

open and close Subroutines

The device driver supports the **/dev/ops0** 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 is used in an **open** call depending on how the device is to be opened. The two types of special file names are:

/dev/ops0	Starts the device driver in normal mode.
/dev/ops0/S	Starts the device driver in serialized mode. As a result, the device driver transmits data in the same order in which it receives the data.

read Subroutine

Can take the form of a **read**, **readx**, **readv**, or **readvx** subroutine. For this call, the device driver copies the data into the buffer specified by the caller.

write Subroutine

Can take the form of a **write, writex**, **writev**, or **writevx** subroutine. For this call, the device driver copies the user data into a kernel buffer and transmits the data.

ioctl subroutine

The Serial Optical Link device driver supports the following **ioctl** operations:

CIO_GET_FASTWRT	Gets attributes needed for the **sol_fastwrt** entry point.
CIO_GET_STAT	Gets device status.
CIO_HALT	Halts the device.
CIO_QUERY	Queries device statistics.
CIO_START	Starts the device.
IOCINFO	I/O character information.
SOL_CHECK_PRID	Checks if a processor ID is connected.
SOL_GET_PRIDS	Gets connected processor IDs.

Error Codes

The following error codes may be returned when accessing the device driver through the **/dev/ops0** special file:

EACCES Indicates access to the device is denied for one of the following reasons:

- A non-privileged user tried to open the device in Diagnostic mode.
- A kernel-mode user attempted a user-mode call.
- A user-mode user attempted a kernel-mode call.

EADDRINUSE Indicates the network ID is in use.

EAGAIN Indicates that the transmit queue is full.

EBUSY Indicates one of the following:

- The device was already initialized.
- There are outstanding opens; unable to terminate.
- The device is already open in Diagnostic mode.
- The maximum number of opens has been exceeded.

EFAULT Indicates that the specified address is not valid.

EINTR Indicates that a system call was interrupted.

EINVAL Indicates that the specified parameter is not valid.

EIO Indicates a general error. If an extension was provided in the call, additional data identifying the cause of the error can be found in the status field.

EMSGSIZE Indicates that the data was too large to fit into the receive buffer and that no *arg* parameter was supplied to provide an alternate means of reporting this error with a status of **CIO_BUF_OVFLW**.

ENETDOWN Indicates that the network is down. The device is unable to process the write.

ENOCONNECT Indicates one of the following:

- The device is not started.
- The processor ID is not connected to the Serial Optical Link subsystem.

ENODEV Indicates that the specified minor number is not valid.

ENOMEM Indicates that the device driver was unable to allocate the required memory.

ENOSPC Indicates the network ID table is full.

EPERM Indicates that the device is open in a mode that does not allow a Diagnostic-mode open request.

Implementation Specifics

This file functions with the Serial Optical Link device driver.

Related Information

Special Files Overview .

The **close** subroutine, **open** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine, **ioctl** subroutine.

Serial Optical Link

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 pair of *control* and *slave* character devices. The slave device provides processes with essentially the same interface as that 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 AIX Version 4, the pty subsystem uses naming conventions similar to those from UNIX System V. There is one node for the control driver, **/dev/ptc**, and a maximum number of *N* nodes for the slave drivers, **/dev/pts/***n*. *N* is configurable at pty configuration and may be changed dynamically by pty reconfiguration, without closing the opened devices.

The control device is set up as a *clone device* whose major device number is the clone device's major number and whose minor device number is the control driver's major number. There is no node in the filesystem for control devices. A control device can be opened only once, but slave devices can be opened several times.

By opening the control device with the **/dev/ptc** special file, 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.

The following ioctl commands apply to pseudo-terminals:

TIOCSTOP	Stops output to a terminal. This is the same as using the Ctrl-S key sequence. No parameters are allowed for this command.
TIOCSTART	Restarts output that was stopped by a **TIOCSTOP** command or by the Ctrl-S key sequence. This is the same as typing the Ctrl-Q key sequence. No parameters are allowed for this command.

TIOCPKT Enables and disables the packet mode. Packet mode is enabled by specifying (by reference) a nonzero 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. The data is 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 defined by the current tty line discipline is Ctrl-S; the start character defined by the line discipline is Ctrl-Q.

TIOCPKT_NOSTOP The start and stop characters are not Ctrl-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** and **rlogind** commands 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 nonzero 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. That command is issued as **UIOCCMD(***Value***)**, where the *Value* parameter specifies 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 can be used with the **UIOCCMD** ioctl operation to probe for the existence of this facility. The zero is not made available for reading by the control 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 implements flow control, rather than input editing, 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, while a write of zero 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 file is part of Base Operating System (BOS) Runtime.

With Berkeley pty subsystems, commands have 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. It is possible to configure the pty subsystem in order to use special files with the BSD pty naming convention:

Control devices	`/dev/pty[p-zA-Z][0-f]`
Slave devices	`/dev/tty[p-zA-Z][0-f]`

These special files are not symbolic links to the AIX special files. The BSD special files are completely separate from the AIX special files. The number of control and slave pair devices using the BSD naming convention is configurable.

In version 3 of the operating system, the pty subsystem used two multiplexed special files, **/dev/ptc** and **/dev/pts**. These special files no longer exist, but the procedure for opening a **pty** device is the same.

Related Information

The **rlogin** command, **rlogind** command.

The **ioctl** subroutine, **ttyname** subroutine.

The Special Files Overview.

tty Subsystem Overview in *AIX General Programming Concepts: Writing and Debugging Programs*.

Understanding TTY Drivers in *AIX General Programming Concepts: Writing and Debugging Programs*.

rcm Special File

Purpose

Provides the application interface to obtain and relinquish the status of a graphics process through the **Rendering Context Manager (RCM)** device driver.

Description

The **rcm** is used by graphics systems such as AIXwindows to obtain a **gsc_handle**. This handle is required in the call to **aixgsc** which is part of the procedure of becoming a graphics process.

Usage Considerations

The **RCM** device driver supports **open**, **close**, and **ioctl** subroutines.

A application uses the GSC_HANDLE ioctl command to get a **gsc_handle** as part of becoming a graphics process. When it closes **rcm**, either normally, or by abnormal termination, the **RCM** releases any displays which it owns. This is implemented as a LFT_REL_DISP ioctl command to the **LFT** device driver.

IOCINFO	Returns **devinfo** structure.
GSC_HANDLE	Returns a gsc_handle.
RCM_SET_DIAG_OWNER	Obtain exclusive use of the display adapter for diagnostics.

Implementation Specifics

The **rcm** special file is part of Base Operating System (BOS) Runtime.

Related Information

lft Special File.

Special Files Overview

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 physical-disk device drivers for physical disks. 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 reserved for system use in managing file systems, paging devices and logical volumes.

The **r** prefix 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

Attention: Data corruption, loss of data, or loss of system integrity (system crashes) 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 the operating system and are solely for system use in managing file systems, paging devices, and logical volumes. They should not be used for other purposes.

open and close Subroutines

The **openx** subroutine provides additional functions to the open sequence. This subroutine requires appropriate permission to execute. Attempting to do so without the proper permission results in a return value of -1, with the **errno** global variable set to **EPERM**.

read and write Subroutines

The **readx** and **writex** subroutines provide for additional parameters affecting the raw data transfer. The *ext* parameter specifies certain options that apply to the request being made. The options are constructed by logically ORing zero or more of the following values.

Note: The following operations can be used only with the **writex** subroutine.

WRITEV	Perform physical write verification on this request.
HWRELOC	Perform hardware relocation of the specified 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.
UNSAFEREL	Perform hardware relocation of the specified block before the block is written. This is done if the drive supports any kind of relocation (safe or unsafe).

ioctl Subroutine

Only one ioctl operation, **IOCINFO**, is defined for all device drivers that use the **ioctl** subroutine. The remaining ioctl operations are all specific to physical-disk devices. Diagnostic mode is not required for the **IOCINFO** operation.

The **IOCINFO** ioctl operation returns a structure for a device type of **DD_DISK**. This structure is defined in the **/usr/include/sys/devinfo.h** file.

Error Codes

In addition to the errors listed for the **ioctl**, **open**, **read**, and **write** subroutines, the following other error codes are also 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** subroutine 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** subroutine 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 **rhdisk** special file is part of Base Operating System (BOS) Runtime.

Files

/dev/hdisk0, **/dev/hdisk1**, ... **/dev/hdisk***n*	Provide block I/O access to the physical volumes (fixed-disk) device driver.

Related Information

Special Files Overview.

The **close** subroutine, **ioctl** subroutine, **lseek** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

Direct Access Storage Device (DASD) Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

SCSI Subsystem Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

scdisk SCSI Device Driver in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

Serial DASD

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 which 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 table shows the names of these special files and their corresponding characteristics:

Tape Drive Special File Characteristics			
Special File Name	**Rewind-on-Close**	**Retension-on-Open**	**Bytes per Inch**
/dev/rmt*	Yes	No	Density setting #1
/dev/rmt*.1	No	No	Density setting #1
/dev/rmt*.2	Yes	Yes	Density setting #1
/dev/rmt*.3	No	Yes	Density setting #1
/dev/rmt*.4	Yes	No	Density setting #2
/dev/rmt*.5	No	No	Density setting #2
/dev/rmt*.6	Yes	Yes	Density setting #2
/dev/rmt*.7	No	Yes	Density setting #2

1. The values of density setting #1 and density setting #2 come from tape drive attributes that can be set using SMIT. Typically density setting #1 is set to the highest possible density for the tape drive while density setting #2 is set to a lower density. However, density settings are not required to follow this pattern.
2. 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.
3. Most tape drives use 512-byte block size. The 8mm tape drive uses a minimum block size of 1024 bytes. Using SMIT to lower the block size, will waste space.

Usage Considerations

Most tape operations are implemented using the **open**, **read**, **write**, and **close** subroutines. However, for diagnostic purposes, the **openx** subroutine is required.

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 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 multitape 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** subroutine 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 a return value of -1, with the **errno** global variable set to **EPERM**.

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 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 block size defined for the magnetic tape device. When the tape drive is in variable block mode, read requests for less than the tape's block size return the number of bytes requested and set the **errno** global variable to a value of 0. In this case, the **readx** subroutine's *Extension* parameter must be set to **TAPE_SHORT_READ**.

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 the 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 the 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 data transfer speed when writing to tape. A write operation sent to tape is returned as complete when the data is transferred to the data buffer of the tape device. The data in the buffer is then written to tape asynchronously. As a result, data-transfer speed increases 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 an EOM condition is encountered.

If an EOM condition is encountered while running in buffered mode, the device attempts to flush the device data buffer. The residual count can exceed the write transfer length in 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 **EIO**. These errors can require the user to run in non-buffered mode.

rmt Special File Considerations: 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 result in a return value of -1, with the **errno** global variable set to **EIO**. Non-reset type errors (that is, media or hardware errors) result in the tape being left positioned where the error occurred.

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.

A signal received by the tape device driver will cause the current command to abort. As a result, the application halts time-consuming commands (for instance, an erase operation) without recycling the drive power or waiting for a timeout to occur.

Use of zero (0) as a block-size parameter indicates the blocksize is of variable length.

ioctl Subroutine

A single ioctl operation, **IOCINFO**, is defined for all device drivers that use the **ioctl** subroutine. 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 **/usr/include/sys/devinfo.h** file.

The STIOCTOP ioctl operation: The **STIOCTOP** ioctl operation provides for command execution operations, such as erase and retension. The parameter to the **ioctl** subroutine using the **STIOCTOP** operation specifies the address of a **stop** structure, as defined in the **/usr/include/sys/tape.h** file.

The operation found in the `st_op` field in the **stop** structure is performed `st_count` times, except for rewind, erase, and retension operations. The available operations are:

STREW Rewind.

STOFFL Rewind and unload the tape. A tape must be inserted before the device can be used again.

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.

STDEOF Disable end-of-file check.

> **Note:** Use of the **STDEOF** command enables an application to write beyond the end of the tape. When disabling end-of-file checking by issuing the **STDEOF** command, it is the responsibility of the application to guard against error conditions that can arise from the use of this command.

Note: Execution of the preceding commands depends 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 **EINVAL**.

Error Codes

In addition to general error codes listed for **ioctl**, **open**, **read**, and **write** subroutines, the following specific error codes 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 an EOM condition or a file mark.
EIO	Could not space forward or reverse `st_count` file marks before encountering an EOM condition.
EMEDIA	The tape device has encountered an unrecoverable media error.
ENOMEM	The number of bytes requested for a read of a variable-length record on tape is less than the actual size (in bytes) of the variable-length record.
ENOTREADY	There is no tape in the drive or the drive is not ready.
ENXIO	A write operation was attempted while the tape was at the EOM.
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 **rmt** SCSI device driver provides further information about implementation specifics.

The **rmt** special file is part of Base Operating System (BOS) Runtime.

Related Information

Special Files Overview .

The **rmt** SCSI Device Driver in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

The **close** subroutine, **ioctl** subroutine,

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). The **/dev/scsi0**, **/dev/scsi1**, ... **/dev/scsi***n* files are the **scsi** special files.

Implementation Specifics

The description of the SCSI Adapter device driver in *AIX Technical Reference: Kernel and Subsystems Volume 1* provides the implementation specifics for the SCSI adapter.

The **scsi** special file is part of Base Operating System (BOS) Runtime.

Related Information

Special Files Overview .

SCSI Subsystem Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

Direct Access Storage Device (DASD) Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

The **scdisk** SCSI Device Driver, and **rmt**

serdasda Special File

Purpose

Provides access to the serial DASD adapter.

Description

The **serdasda** special file provides an interface to an attached Serial DASD adapter. This special file should not be opened directly by application programs (with the exception of diagnostics applications). The **/dev/serdasda0**, **/dev/serdasda1**, ... **/dev/serdasda***n* files are the **serdasda** special files.

Usage Considerations

The description of the serial DASD subsystem device driver in *AIX Technical Reference: Kernel and Subsystems Volume 1* provides information on using the Serial DASD adapter.

Implementation Specifics

The description of the Serial DASD subsystem device driver in *AIX Technical Reference: Kernel and Subsystems Volume 1* provides information on implementation specifics for the Serial DASD adapter.

The **serdasda** special file is part of Base Operating System (BOS) Runtime.

Related Information

The Direct Access Storage Device (DASD) Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

Device-Dependent Subroutines for Serial DASD Adapter Operations in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

Error Conditions for Serial DASD Subroutines in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

Special Files Overview .

serdasdc Special File

Purpose

Provides access to the serial DASD subsystem controllers.

Description

The **serdasdc** special file provides an interface to an attached serial DASD subsystem controllers. This special file should not be opened directly by application programs (with the exception of diagnostics applications).The **/dev/serdasdc0**, **/dev/serdasdc1**, ... **/dev/serdasdc***n* files provide access to the serial DASD subsystem controllers.

Usage Considerations

The description of the Serial DASD subsystem device driver in *AIX Technical Reference: Kernel and Subsystems Volume 1* provides information on using the Serial DASD controllers.

Implementation Specifics

The description of the Serial DASD subsystem device driver in *AIX Technical Reference: Kernel and Subsystems Volume 1* provides information on implementation specifics for the Serial DASD controllers.

The **serdasdc** special file is part of Base Operating System (BOS) Runtime.

Related Information

The Direct Access Storage Device (DASD) Overview in *AIX Kernel Extensions and Device Support Programming Concepts*.

Device-D ependent Subroutines for Serial DASD Controller Operations in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

Error Conditions for Serial DASD Subroutines in *AIX Technical Reference: Kernel and Subsystems Volume 1*.

Special Files Overview .

tablet Special File

Purpose

Provides access to the tablet.

Description

The tablet special file is the application interface to the tablet. It provides the applications with the capability of receiving input from the tablet and it allows the application to change the sampling rate, dead zones, origin, resolution, and conversion mode.

Configuration

There are no user commands to change the configuration of the tablet device. Applications may use **ioctl** commands to modify the configuration but these modifications are effective only until the tablet is closed.

Usage Considerations

The **open** subroutine call specifying the **tablet** special file is processed normally except that the *Oflag* and *Mode* parameters are ignored. The open request is rejected if the special file is already opened or if a kernel extension attempts to open the special file. All tablet inputs are flushed following an **open** subroutine call until an input ring is established. The tablet device is reset to the default configuration when an open request is made.

The **tablet** special file does not support the **read** or **write** subroutine calls. Instead, input data is obtained from the tablet through the input ring. The **read** and **write** subroutine calls behave the same as **read** or **write** subroutine calls to the **/dev/null** file.

The **tablet** special file supports the following functions with **ioctl** subroutines:

IOCINFO	Returns **devinfo** structure.
TABCONVERSION	Sets tablet conversion mode.
TABDEADZONE	Sets tablet dead zones.
TABFLUSH	Flushes input ring.
TABORIGIN	Sets tablet origin.
TABQUERYID	Queries tablet device identifier.
TABREGRING	Registers input ring.
TABRESOLUTION	Sets resolution.
TABSAMPELRATE	Sets sample rate.

Error Codes

The error codes can be found in the **/usr/include/sys/errno.h** file.

EFAULT	Indicates insufficient authority to access address or invalid address.
EIO	Indicates an I/O error.
ENOMEM	Indicates insufficient memory for required paging operation.
ENOSPC	Indicates insufficient file system or paging space.
EINVAL	Indicates an invalid argument.
EINTR	Indicates the request was interrupted by signal.
EPERM	Indicates a permanent error occurred.
EBUSY	Indicates the device is busy.
ENXIO	Indicates an unsupported device number was specified.
ENODEV	Indicates an unsupported device or device type mismatch.
EACCES	Indicates **open** is not allowed.

Implementation Specifics

The **tablet** special file is part of Base Operating System (BOS) Runtime.

Files

/usr/include/sys/inputdd.h	Contains declarations for ioctl commands and input ring report format.

Related Information

LFT Input Devices in *AIX Version 4.3 Kernel Extensions and Device Support Programming Concepts*.

The **close** subroutine, **ioctl** subroutine, **open** subroutine, **read** subroutine, **write** subroutine.

The **dials** special file, **GIO** special file, **kbd** special file, **lpfk** special file, **mouse** special file.

Special Files Overview

tmscsi Special File

Purpose

Provides access to the SCSI **tmscsi** device driver.

Description

The **tmscsi** special file provides an interface to allow processor-to-processor data transfer using the SCSI **send** command. This single device driver handles both SCSI initiator and SCSI target mode roles.

The user accesses the data transfer functions through the special files **/dev/tmscsi0.***xx*, **/dev/tmscsi1.***xx*, ... These are all character special files. The *xx* variable can be either **im**, initiator-mode interface, or **tm**, target-mode interface. The initiator-mode interface transmits data, and the target-mode interface receives data.

The least significant bit of the minor device number indicates to the device driver which mode interface is selected by the caller. When the least significant bit of the minor device number is set to 1, the target-mode interface is selected. When the least significant bit is set to 0, the initiator-mode interface is selected.

When the caller opens the initiator-mode special file, a logical path is established allowing data to be transmitted. The **write**, **writex**, **writev**, or **writevx** subroutine initiates data transmission for a user-mode caller, and the **fp_write** or **fp_rwuio** kernel services initiate data transmission for a kernel-mode caller. The SCSI target-mode device driver then builds a SCSI **send** command to describe the transfer, and the data is sent to the device. Once the write entry point returns, the calling program can access the transmit buffer.

When the caller opens the target-mode special file, a logical path is established allowing data to be received. The **read**, **readx**, **readv**, or **readvx** subroutine initiates data reception for a user-mode caller, and the **fp_read** or **fp_rwuio** kernel service initiates data reception for a kernel-mode caller. The SCSI target-mode device driver then returns data received for the application.

Implementation Specifics

The SCSI **tmscsi** device driver provides further information about implementation specifics.

The **tmscsi** special file is part of Base Operating System (BOS) Runtime.

Special Files Overview.

Note: This operation is not supported by all SCSI I/O controllers.

Related Information

The **close** subroutine, **open** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine.

SCSI Target-Mode

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 that 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.

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 depends upon how the device is to be opened. The three 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.

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.

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.

ioctl Subroutine

The token-ring device handler supports the following ioctl operations:

CIO_GET_STAT	Returns current adapter and device handler status.
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 **/usr/include/sys/comio.h** and **/usr/include/sys/tokuser.h** files.
CIO_START	Starts a session and registers a network ID.
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 **/usr/include/sys/devinfo.h** file.
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_QVPD	Returns adapter vital product data.
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 not valid.
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

Special Files Overview .

trace Special File

Purpose

Supports event tracing.

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. Write trace events 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 Software Trace Service Aids package.

Related Information

The **trcgenk** kernel service.

tty Special File

Purpose

Supports the controlling terminal interface.

Description

For each process, the **/dev/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 if 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** subroutine. 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 by using the **setsid** subroutine. 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 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 AIX compatibility mode.

The **tty** special file is part of Base Operating System (BOS) Runtime.

Related Information

The **fork** subroutine, **open** subroutine, **setsid** subroutine.

Special Files Overview .

The tty Subsystem

x25sn Special File

Purpose

Provides access to the X.25 Interface Co-Processor/2 adapters by way of the X.25 Interface Co-Processor/2 device handler.

Description

The **x25s***n* character special file provides access to the X.25 Interface Co-Processor/2 device handler, which provides access to a X.25 packet switching network. The device handler supports up to four X.25 Interface Co-Processor/2 adapters.

Usage Considerations

When accessing the X.25 Interface Co-Processor/2 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 X.25 Interface Co-Processor/2 device handler provides specific support for the **open**, **close**, **read**, **write**, and **ioctl** subroutines.

open and close Subroutines

The device handler supports the **/dev/x25s***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. For each of the following types of special files, the value *n* is 0 <= *n* <= 7:

/dev/x25s*n*
: Starts the device handler on the next available port.

/dev/x25s*n***/D**
: Opens the device handler for the specified port in Diagnostic mode.

/dev/x25s*n***/M**
: Opens the device handler for reading and writing data to the monitor facilities on the X.25 Interface Co-Processor/2.

/dev/x25s*n***/R**
: Opens the device handler for updating the routing table.

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.

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 network.

ioctl Subroutine

The device handler supports the following ioctl operations:

CIO_DNLD	Downloads a task.
CIO_GET_STAT	Gets device statistics.
CIO_HALT	Halts a session.
CIO_QUERY	Queries a device.
CIO_START	Starts a session.
IOCINFO	Identifies a device.
X25_ADD_ROUTER_ID	Adds a router ID.
X25_COUNTER_GET	Gets a counter.
X25_COUNTER_READ	Reads the contents of a counter.
X25_COUNTER_REMOVE	Removes a counter from the system.
X25_COUNTER_WAIT	Waits for the contents of counters to change.
X25_DELETE_ROUTER_ID	Deletes a router ID.
X25_DIAG_IO_READ	Reads to an I/O register on the X.25 Interface Co-Processor/2.
X25_DIAG_IO_WRITE	Writes to an I/O register on the X.25 Interface Co-Processor/2.
X25_DIAG_MEM_READ	Reads memory from the X.25 Interface Co-Processor/2 into a user's buffer.
X25_DIAG_MEM_WRITE	Writes memory to the X.25 Interface Co-Processor/2 from a user's buffer.
X25_DIAG_TASK	Provides the means to download the diagnostics task on to the card.
X25_LINK_CONNECT	Connects a link.
X25_LINK_DISCONNECT	Disconnects a link.
X25_LINK_STATUS	Returns the status of the link.
X25_LOCAL_BUSY	Enables or disables receiving of data packets on a port.
X25_REJECT	Rejects a call.
X25_QUERY_ROUTER_ID	Queries a router ID.
X25_QUERY_SESSION	Queries a session.

Error Conditions

The following error conditions may be returned when accessing the device handler through the **/dev/x25s***n* special file:

EACCES	Indicates that the call application does not have the required authority.
EAGAIN	Indicates there were no packets to be read or the transmit queue is full, and the device was opened with the **DNDELAY** flag set.
EBUSY	Indicates that the device was busy and could not accept the operation.
EFAULT	Indicates that an invalid address was specified.
EIDRM	Indicates that the counter has been removed.
EINTR	Indicates that the subroutine call was interrupted.
EINVAL	Indicates that an invalid parameter was passed to one of the subroutine calls.
EIO	Indicates that an error has occurred. The status field in the status-control block contains more information.
EMSGSIZE	Indicates that the data to be given to the user was greater than the length of the buffer specified. The data in the buffer is truncated.
ENOBUFS	Indicates that no buffers are available.
ENODEV	Indicates that the device requested does not exist.
ENOMEM	Indicates that the X.25 device handler was unable to allocate space required for the open.
ENOSPC	Indicates that there are no counters available to allocate.
ENXIO	Indicates that the device was not completely configured. Initial configuration must be completed before any starts can be issued.
EPERM	Indicates the user does not have permission to perform the requested operation.

Related Information

Special Files Overview.

The **close** subroutine, **open** subroutine, **read** or **readx** subroutine, **write** or **writex** subroutine.

Chapter 4. Header Files

Information that is needed by several different files or functions is collected into a header file. A header file contains C-language definitions and structures. Centralizing 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 define the following functions:

- Structures of certain files and subroutines
- Type definition (typedef) synonyms for data types
- System parameters or implementation characteristics
- 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 operating system; however, the suffix is not required for user-generated header files.

Note: Several of the header files provided with the operating system end with **.inc** (include file).

Additional header files are provided with the operating system. Most of these can be found in either the **/usr/include** directory or the **/usr/include/sys** directory. Use the **pg** command to view the contents of a header file.

More information about the following header files is provided in this documentation:

a.out.h Defines the structure of the standard **a.out** file. **acct.h** Describes the format of the records in the system accounting files. **ar.h** Describes the format of an archive file. **audit.h** Defines values used by the auditing system as well as the structure of a bin. **core.h** Describes the structures created as a result of a core dump. **dirent.h** Describes the format of a file system-independent directory entry. **eucioctl.h** Defines ioctl operations and data types for handling EUC code sets. **fcntl.h** Defines values for the **fcntl** and **open** subroutines. **filsys.h** Contains the format of a file system logical volume. **flock.h** Defines the file control options. **fullstat.h** Describes the data structure returned by the **fullstat** and **ffullstat** subroutines. **iconv.h** Defines types, macros, and subroutines for character code set conversion. **ipc.h** Defines structures used by the subroutines that perform interprocess communications operations. **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. **msg.h** Defines structures used by the subroutines that perform message queueing operations. **param.h** Defines certain hardware-dependent parameters. **poll.h** Defines the **pollfd** structure used by the **poll** subroutine. **sem.h** Defines the structures that are used by subroutines that perform semaphore operations. **sgtty.h** Defines structures used by the Berkeley terminal interface. **shm.h** Defines structures used by the subroutines that perform shared memory operations. **spc.h** Defines external interfaces provided by the System Resource Controller (SRC) subroutines. **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. **statvfs.h** Describes the structure of the statistics that are returned by the **statvfs** subroutines and **fsatvfs** subroutines. **systemcfg.h** Defines the **_system_configuration** structure. **tar.h** Defines flags used in the **tar** archive header. **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. **termiox.h** Defines the structure of the **termiox** file, which provides the extended 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 X.25 application programming interface.

3270 Host Connection Program (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 g32hfile.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** Contains common HCON API key value definitions for the Pascal language. **HCON g32types.inc** Defines HCON API data types for the Pascal language.

Related Information

The **pg** command.

File Formats Overview defines and describes file formats in general and lists file formats discussed in this documentation.

Special Files Overview defines and describes special files in general and lists special files discussed in this documentation.

List of Major Control Block Header Files

The Base Operating System constants and control block structure definitions are contained in header files in the **/usr/include** and **/usr/include/sys** directories. The major constants and control blocks and their corresponding header files are:

/usr/include/a.out.h	Common Object File Format (COFF) structures
/usr/include/core.h	An include file for the **/usr/include/sys/core.h** header file
/usr/include/errno.h	An include file for the **/usr/include/sys/errno.h** header file
/usr/include/lvmrec.h	LVM record structure
/usr/include/sgtty.h	Line discipline structures and constants for Berkeley compatibility
/usr/include/signal.h	An include file for the **/usr/include/sys/signal.h** header file
/usr/include/termio.h	An include file for the **/usr/include/sys/termio.h** header file
/usr/include/termios.h	POSIX line-discipline structures and constants
/usr/include/xcoff.h	Extended Common Object File Format structures
/usr/include/sys/acct.h	Accounting structures
/usr/include/sys/badisk.h	Bus-attached-disk structures
/usr/include/sys/bbdir.h	Bad-block directory structure
/usr/include/sys/bootrecord.h	Boot record structure
/usr/include/sys/buf.h	Buffer header structures
/usr/include/sys/cdrom.h	CD-ROM structures
/usr/include/sys/cfgodm.h	Configuration object class structures
/usr/include/sys/configrec.h	Disk configuration record structure
/usr/include/sys/core.h	Core dump structure
/usr/include/sys/debug.h	Traceback table or procedure-end table
/usr/include/sys/device.h	Device switch table
/usr/include/sys/deviceq.h	Device queue-management structures
/usr/include/sys/devinfo.h	Device information structures
/usr/include/sys/dir.h	Directory entry structures
/usr/include/sys/display.h	Virtual display driver structures

/usr/include/sys/dump.h	Component dump table structure
/usr/include/sys/entuser.h	Ethernet device driver structures
/usr/include/sys/errids.h	Error-log record identifiers
/usr/include/sys/errno.h	Error codes
/usr/include/sys/fd.h	Diskette device driver structures
/usr/include/sys/file.h	File structure
/usr/include/sys/fstypes.h	File-system parameter table
/usr/include/sys/hd_psn.h	Layout of reserved space on the disk
/usr/include/sys/ide.h	IDE device driver structures
/usr/include/sys/inode.h	I-node structures
/usr/include/sys/intr.h	Interrupt handler structures
/usr/include/sys/ipc.h	Interprocess Communications (IPC) structures
/usr/include/sys/iplcb.h	Initial Program Load (IPL) control block structure
/usr/include/sys/ldr.h	Loader structures and constants
/usr/include/sys/low.h	Kernel Page 0 definition
/usr/include/sys/machine.h	Machine dependent control registers
/usr/include/sys/mbuf.h	Memory buffer structures
/usr/include/sys/mdio.h	Machine device driver structures
/usr/include/sys/mount.h	Mount structures
/usr/include/sys/mpqp.h	Multiprotocol Quad Port (MPQP) device-driver structures
/usr/include/sys/msg.h	Message queue structures
/usr/include/sys/mstsave.h	Machine State Save Area structures
/usr/include/sys/param.h	Process management constants
/usr/include/sys/pri.h	Constants for process priorities
/usr/include/sys/proc.h	Process table structure
/usr/include/sys/pseg.h	Process private segment layout
/usr/include/sys/reg.h	Machine-dependent registers
/usr/include/sys/scdisk.h	SCSI-disk device driver structures
/usr/include/sys/scsi.h	SCSI device driver structures
/usr/include/sys/seg.h	Memory management constants

/usr/include/sys/sem.h	Semaphore structures
/usr/include/sys/shm.h	Shared-memory facility structures
/usr/include/sys/signal.h	Signal structures and constants
/usr/include/sys/socketvar.h	Sockets structures
/usr/include/sys/stat.h	File status structure
/usr/include/sys/systm.h	System global declarations
/usr/include/sys/termio.h	Line discipline structures and constants for AIX Version 2 compatibility
/usr/include/sys/timer.h	Timer structures
/usr/include/sys/tokuser.h	Token-ring device handler structures
/usr/include/sys/trchkid.h	Trace hook IDs
/usr/include/sys/user.h	User structure or user area (ublock)
/usr/include/sys/utsname.h	UTSNAME structure (system name, node ID, machine ID)
/usr/include/sys/var.h	Runtime system parameter structure
/usr/include/sys/vfs.h	Virtual file system structures
/usr/include/sys/vnode.h	Virtual i-node (v-node) structures
/usr/include/sys/xcoff.h	Extended Common Object File Format structures
/usr/include/sys/xmalloc.h	Heap structure
/usr/include/sys/xmem.h	

Options and Flags for HCON File Transfer Header Files

The **fxfer.h** and **fxfer.inc** header files have the **fxc** structure in common. This structure defines options used by both the C and Pascal header file.

C and Pascal Options

The options in the C structures and Pascal record declarations for the File Transfer Program Interface are:

`f_logonid`
: Contains the host login ID string. This value should contain the host login ID, the AUTOLOG node ID, and two optional AUTOLOG parameters, all separated by commas. This list is passed to the automatic login procedure.

  At run time, the operator is asked to enter a password. The host login session is maintained for subsequent file transfers, eliminating the need to log in again. The file transfer wait period in the HCON session profile variable determines the length of time the login session is maintained.

`f_inputfld`
: Specifies a host input field. This option enables the user to place host file transfer program (**IND$FILE**) options on the host file-transfer program command line. It also allows the user to place comments within the command. This option is valid only for CICS and VSE hosts.

`f_aix_codepg`
: Specifies an alternate code set to use for an ASCII-to-EBCDIC or EBCDIC-to-ASCII translation. If this field is null, the ASCII code set is extracted from the system locale.

FXC_APPND
: Appends the file specified by the source file to the destination file if the destination file exists when the **FXC_APPND** flag is set in the `fxc_opts.f_flags` field. This option is ignored if the destination file does not exist. This option is not valid when uploading to a CICS or VSE host.

FXC_CICS
: Specifies the host as CICS/VS when the **FXC_CICS** flag is set. The user must specify the correct host operating system. The file-transfer program does not distinguish between the four host operating systems.

FXC_CMS
: Specifies the host as VM/CMS when the **FXC_CMS** flag is set in the `fxc_opts.f_flags` field. The user must specify the correct host operating system. The file-transfer program does not distinguish between the four host operating systems.

FXC_DOWN
: Downloads the file from a host file to a file when the **FXC_DOWN** flag is set in the `fxc_opts.f_flags` field.

FXC_QUEUE Executes the file transfer asynchronously as a background process when set in the `fxc_opts.f_flags` field. If any file transfers have not completed, the current transfer request is queued. If this option is not specified, the file-transfer operation is synchronous.

FXC_REPL Replaces an existing file on the host (upload) or replaces an existing file (download) when the **FXC_REPL** flag is set in the `fxc_opts.f_flags` field. When uploading to a CICS host, this option is the default.

FXC_TSO Specifies the host as MVS/TSO (Multiple Virtual Storage/Time Sharing Option) when set in the `fxc_opts.f_flags` field. The user must specify the correct host operating system. The file-transfer program does not distinguish between the four host operating systems.

FXC_TNL Translates EBCDIC to ASCII when downloading files, if set in the `fxc_opts.f_flags` field. During uploading, the **FXC_TNL** option translates ASCII to EBCDIC. This option assumes the file is a text file and is used when transferring formatted text files. The default is no translation. The new-line character is the line delimiter.

FXC_TCRLF Performs the same function as the **FXC_TNL** option when set in the `fxc_opts.f_flags` field, except that the line delimiter is the carriage return/line-feed (CR-LF) character sequence. This option is used to translate PC-DOS files. A PC-DOS end-of-file character is inserted at the end of the downloaded file.

> **Note:** If neither the **FXC_TNL** nor the **FXC_TCRLF** option is specified, the file transfer assumes no translation and transfers the data in binary form. When transferring a binary file to the host, the host file-transfer program defaults the host file to a fixed record format. If the user wishes not to have the host file padded with blanks at the end, the **FXC_VAR** option should be specified to delineate a variable record format.

FXC_UP Uploads the file from the operating system file to the host file when set in the `fxc_opts.f_flags` field.

FXC_VSE Specifies the host as VSE/ESA (Virtual Storage Extended/Enterprise Systems Architecture) or VSE/SP (VSE/System Product) when set in the `fxc_opts.f_flags` field. The user must specify the correct host operating system. The file transfer program does not distinguish between the four operating systems.

Host File Flags

The following flags specify host file characteristics. They can be used only to upload files, with the exception of the **FXC_FIXED** option, which can be used when downloading from a VSE host.

f_blksize Specifies the nonzero block size of the host data set. This option is only used in the MVS/TSO environment. For new files, the default is the logical record length. This flag is ignored if the file is being appended.

f_lrecl Specifies the nonzero logical record length of the host file. For new files, the default is 80. For variable-length records, the `f_lrecl` field is the maximum size of the record. This field is ignored if the file is being appended. If using this option while uploading to a CICS or VSE host, the **FXC_FIXED** option flag must also be specified.

FXC_FIXED Specifies fixed-length records when set in the `fxc_opts.f_flags` field. This is the default if none of the following flags are set: **FXC_VAR**, **FXC_TNL**, and **FXC_TCRLF**. This flag is ignored if the file is being appended. If specifying this option while uploading to a CICS or VSE host, either the **FXC_TNL** or the **FXC_TCRLF** option flag must be specified. If this option is specified when downloading from a VSE host, all trailing blanks are downloaded. The default option when downloading a translatable file from a VSE host causes all trailing blanks to be deleted.

FXC_UNDEF Specifies records of undefined length when set in the `fxc_opts.f_flags` field. This option can only be used in the MVS/TSO environment and is ignored if the file is being appended.

FXC_VAR Specifies variable-length records when set in the `fxc_opts.f_flags` field. This is the default if the **FXC_FIXED** flag is not set and either the **FXC_TNL** or the **FXC_TCRLF** flag is set. This flag is ignored if the file is being appended.

s_space Specifies the non-zero number of units of space to be allocated for a new data set. This option can only be used in the MVS/TSO environment. The `s_space` field has the following optional subfields:

s_increment Specifies the number of units of space to be added to the data set each time the previously allocated space is filled.

s_unit Specifies the unit of space. A value of **FXC_TRACKS** indicates the unit of allocation is tracks. A value of **FXC_CYLINDERS** indicates the unit of allocation is cylinders. Otherwise, the `s_space` field specifies the average block size (in bytes) of the records to be written to the data set. If the `s_space` field has a value of zero, the default unit of allocation is the value specified by the `f_blksize` field. If the `f_blksize` field is not specified, the host file-transfer program uses the default value of 80.

Related Information

The **cfxfer** function, **fxfer** function, **g32_fxfer** function.

The **fxfer** command.

HCON Host Logon Procedures in *3270 Host Connection Program 2.1 and 1.3.3 for AIX: Guide and Reference*.

HCON File Transfers in *3270 Host Connection Program 2.1 and 1.3.3 for AIX: Guide and Reference*.

dirent.h File

Purpose

Describes the format of a file system-independent directory entry.

Description

The **/usr/include/dirent.h** file describes the format of a directory entry without reference to the type of underlying system.

The **dirent** structure, defined in the **dirent.h** file, is used for 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 contains the following fields 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 the **PATH_MAX** constant, which specifies the maximum number of bytes in the full path name of a file, not including the terminating null byte.)

The value of the **_D_NAME_MAX** constant is specific to each type of filesystem type. It can be determined by using the **pathconf** or **fpathconf** subroutine.

The size of a **dirent** structure depends on the number of bytes in the file name.

Implementation Specifics

The **_DNAME_MAX** and **PATH_MAX** constants specify maximum file names and path names, respectively, across all types of file systems. The constants defined by a particular file system are applicable only to that file system. 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 Base Operating System (BOS) Runtime.

Related Information

The **dir** file, **sys/types.h** file.

The **pathconf** or **fpathconf** subroutine.

Understanding JFS i-nodes in *AIX General Programming Concepts: Writing and Debugging Programs* explains how the operating system uses i-nodes.

The Header Files Overview defines header files, describes how they are used, and lists several header files for which information is provided.

dlfcn.h File

Purpose

Describes dynamic linking.

Syntax

#include <dlfcn.h>

Description

The **<dlfcn.h>** header defines at least the following macros for use in the construction of a dlopen mode argument:

RTLD_LAZY	Relocations are performed at an implementation-dependent time.
RTLD_NOW	Relocations are performed when the object is loaded.
RTLD_GLOBAL	All symbols are available for relocation processing of other modules
RTLD_LOCAL	All symbols are not made available for relocation processing by other modules.

The header **<dlfcn.h>** declares the following functions, which may also be defined as macros:

```
void    *dlopen(const char *, int);
void    *dlsym(void *, const char *);
int      dlclose(void *);
char    *dlerror(void);
```

Related Information

The **dlopen**, **dlclose**, **dlsym**, **dlerror** subroutines.

euciocti.h File

Purpose

Defines ioctl operations and data types for handling EUC code sets.

Description

The **eucioctl.h** file contains information used for handling Extended UNIX Code (EUC) multibyte code sets. It consists of ioctl operations and the related data structure.

The **eucioc** structure contains the following fields:

`eucw[4]`	Specifies the memory width of the code set. It indicates the number of bytes used to store the multibyte characters of each of the four classes.
`scrw[4]`	Specifies the screen width of the code set. It indicates the number of columns used to display the multibyte characters of each of the four classes.

This structure is used in the following ioctl operations:

EUC_WGET	Returns the EUC character widths. The **eucioc** structure is filled with the memory and screen widths of the current EUC code set.
EUC_WSET	Sets the EUC character widths. The **eucioc** structure is used to set the memory and screen widths of the current EUC code set.

Implementation Specifics

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

Related Information

The **ioctl** subroutine.

tty Subsystem Overview in *AIX General Programming Concepts: Writing and Debugging Programs*.

fcntl.h File

Purpose

Defines file control options.

Description

The **/usr/include/fcntl.h** 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 information.

Flag Values for open Subroutine

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_DIRECT	Open for Direct I/O
O_EXCL	Exclusive open **Note:** The **O_EXCL** flag is not fully supported for Network File Systems (NFS). The NFS protocol does not guarantee the designed function of the **O_EXCL** flag.
O_NOCTTY	Do not assign a controlling terminal
O_RSHARE	Read shared open
O_NSHARE	Read shared open

File Access Mode Mask

The **O_ACCMODE** mask is used to determine the file access mode.

File Status flags for open and fcntl Subroutines

The following file status flags are accessible to both the **open** and **fcntl** subroutines:

O_NONBLOCK	POSIX nonblocking I/O
FNONBLOCK	POSIX nonblocking 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
O_DSYNC	Synchronous write option (file data only).
FDATASYNC	Synchronous write option (file data only).
O_RSYNC	Synchronous file attributes on read.
FREADSYNC	Synchronous file attributes on read.
FASYNC	Asynchronous I/O
O_NDELAY	Nonblocking I/O
FNDELAY	Nonblocking I/O
O_LARGEFILE	Access to large files enabled (AIX versions 4.2 and later)

File Status Flags for open Subroutine

The following file status flags are accessible to the **open** subroutine:

O_DEFER	Deferred update
O_DELAY	Open with delay
O_DIRECT	Open for Direct I/O

File Descriptor Flags for fcntl Subroutine

The following file descriptor flag is accessible to the **fcntl** subroutine:

FD_CLOEXEC	Close this file during an exec.

File flag values corresponding to file access modes are as follows:

FREAD	File is open for read.
FWRITE	File is open for write.

Notes:

1. The **FREAD** and **FWRITE** flags cannot be used unless the **_KERNEL** flag has been defined.

2. The **ldfcn.h** file also assigns values to the **FREAD** and **FWRITE** options. If you use the **ldfcn.h** and **fcntl.h** files together, directly or indirectly, you should use the **#undef** statement on the **FREAD** and **FWRITE** options of one of the header files. If you do not, the compiler will return a warning about using duplicate definitions.

Command Values for fcntl Subroutine

The *Command* values for the **fcntl** subroutine (that is, for **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 status flags and file access modes.
F_SETFL	Set the file flags.
F_GETLK	Return information about an existing file lock.
F_GETLK64	Return information about an existing file lock (AIX versions 4.2 and later) .
F_SETLK	Set or clear a file lock.
F_SETLK64	Set or clear a file lock (AIX versions 4.2 and later) .
F_SETLKW	Set or clear a file lock and wait if blocked.
F_SETLKW64	Set or clear a file lock and wait if blocked (AIX versions 4.2 and later) .
F_GETOWN	Get the descriptor owner.
F_SETOWN	Set the descriptor owner.

Related Information

The **fcntl** subroutine, **open**, **openx**, or **creat** subroutine.

The **sys/types.h** file, **unistd.h** file.

The Header Files Overview

filsys.h File

Purpose

Contains the format of a Journaled File System (JFS) logical volume.

Syntax

#include <sys/filsys.h>

Description

The **filsys.h** file contains the format of a JFS file system. A JFS file system has a common format for vital information and is divided into a number of fixed-sized units, or fragments. Fragments serve as the basic unit of file system disk space allocation and can be smaller than the file system logical block size, which is 4096 bytes. The file system superblock records the logical block size and fragment size, as well as the size of the entire file system.

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

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 Fragment allocation bit map
INODEX_I	i-node extensions
INODEXMAP_I	Allocation map for i-node extensions

The first 4096 bytes of the file system are unused and available to contain a bootstrap program or other information. The second 4096 bytes of the file system are used to hold the file system superblock. The structure of a JFS superblock follows:

```
/* The following disk-blocks are formatted or reserved:
 *
 *      ipl block 0 - not changed by filesystem.
 *
 *      superblocks at  1 x 4096 (primary superblock) and  31 x
 *      4096 (secondary superblock). the secondary super-block
 *      location is likely to be on a different disk-surface than
```

```
 *      the primary super-block. both structures are allocated as
 *      fragments in ".superblock".
 *
 *      fragments for .inodes according to BSD layout. each
 *      allocation group contains a fixed number of disk inodes.
 *      for fsv3 file systems, each allocation group contains one
 *      inode per 4096 byte fragment of the allocation group,
 *      with the number of fragments within each group described
 *      by the s_agsize field of the superblock. for fsv3p file
 *      systems, the number of inodes per group is described by
 *      the s_iagsize field of the superblock and may be less
 *      than or greater than the number of fragments per group.
 *      for these file systems, s_agsize describes the number of
 *      s_fragsize fragments contained within each allocation
 *      group.
 *
 *      the first allocation group inodes starts at 32 x
 *      4096 bytes and consumes consecutive fragments sufficient
 *      to hold the group's inodes. the inode fragments for all
 *      other allocation groups start in the first fragments of
 *      the allocation group and continue in consecutive
 *      fragments sufficient to hold the group's inodes.
 *
 *      other fragments 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 the
 * second 4096 bytes of the file system.
 * the secondary super-block is not used except for disaster
 * recovery.
*/
 struct superblock
 {
   char s_magic[4];     /* magic number */
   char s_flag[4];      /* flag word (see below) */
   int  s_agsize;       /* fragments per allocation group */
   int  s_logserial;  /* serial number of log when fs mounted */
   daddr_t s_fsize;     /* size (in 512 bytes) of entire fs */
   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      */

  /* more persistent
information              &
nbsp;               &
nbsp;*/
  int s_version;    /* version
number
           */
  int s_fragsize;    /* fragment size in bytes (fsv3p only)    */
  int s_iagsize;     /* disk inodes per alloc grp (fsv3p only) */
  int s_compress;    /* > 0 if data compression                */

};

    /* Version 3 fs magic number  */
    #define fsv3magic  "\102\041\207\145"
    /* Version 3p fs magic number */
    #define fsv3pmagic "\145\207\041\102"
    /* Version 3p version number  */
    #define fsv3pvers  1
```

The path name of this file is **/usr/include/jfs/filsys.h**. But, if the **/usr/include/sys/filsys.h** file is included, the **/usr/include/jfs/filsys.h** file is included by default.

The fields of the AIX superblock structure have the following functions:

`s_fname`	Specifies the name of the file system.
`s_fpack`	Specifies the name of the volume on which the file system resides.
`s_fsize`	Specifies the entire file system size in 512-byte units.
`s_bsize`	Specifies the file-system logical block size in bytes.

`s_fragsize` Specifies the file system fragment size and is only valid for fsv3p file systems. For fsv3 file systems, the file-system fragment size is logically defined as the file-system logical block size.

`s_agsize` Specifies the number of fragments per file system allocation group. For fsv3 file systems, this field also specifies the number of disk i-nodes per file system allocation group.

`s_iagsize` Specifies the number of disk i-nodes per file system allocation group for fsv3p file systems. The `s_iagsize` field is only valid for fsv3p file systems.

`s_magic` Specifies the file-system magic number and is used to validate file systems. The *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 fsv3 superblock, use a condition similar to:

```
if (strncmp(sp->s_magic,fsv3magic,4) == 0)
```

For fsv3p file systems, superblock validation is made by checking both the `s_magic` and `s_version` fields.

`s_version` Specifies the file-system version number and is only valid for fsv3p file systems. To check for a valid fsv3p superblock, use a condition similar to:

```
if (strncmp(sp->s_magic,fsv3pmagic,4) == 0 &&
    sp->s_version == fsv3pvers)
```

`s_logdev` Specifies the device ID of the file system log device.

`s_logserial` Records the serial number of the log device at the time the file system was last mounted as modifiable.

`s_fmod` Contains 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 equal to nonzero. A file system whose `s_fmod` field is equal to 0 is very likely to be clean, and a file system whose `s_fmod` field is equal to 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 dirty.

If you only 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, the flag 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.

`s_ronly` Contains 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.

`s_time` Specifies the last time the superblock of the file system was changed (in seconds since 00:00 Jan. 1, 1970 (GMT)).

Implementation Specifics

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

Related Information

The **param.h** file format.

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

The File Systems Overview for System Management in *AIX Version 4.3 System Management Concepts: Operating System and Devices* explains file system types, management, and structure.

The Mounting Overview in *AIX Version 4.3 System Management Concepts: Operating System and Devices* explains mounting files and directories, mount points, and automatic mounts.

The Logical Volume Storage Overview in *AIX Version 4.3 System Management Concepts: Operating System and Devices* explains the Logical Volume Manager, physical volumes, logical volumes, volume

flock.h File

Purpose

Defines file control options.

Description

The **flock** structure in the **/usr/include/sys/flock.h** 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_SETLK** or **F_SETLKW**, 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_GETLK**, 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** 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.

Although the **flock** structure is used by application programs to make file lock requests, the extended **flock** structure, **struct eflock**, is used internally by the kernel. The **eflock** structure is identical to the **flock** structure in that it has the same fields. The differences are that the l_len and l_start fields are 64 bit integers.

The **flock64** structure (AIX versions 4.2 and later) in the **/usr/include/sys/flock.h** 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_SETLK** or **F_SETLKW**, 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_GETLK**, 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** file.

`l_start`
: Defines the relative offset in bytes, measured from the starting point in the `l_whence` field. This field is of the type `off64_t`.

`l_len`
: Specifies the number of consecutive bytes to be locked. This field is of the type `off64_t`.

`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.

Implementation Specifics

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

Related Information

The **unistd.h** file.

The **fcntl** subroutine, **lockfx**, **lock**, or **flock** subroutine, **open**, **openx**, or **creat** subroutine.

Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

fullstat.h File

Purpose

Defines the data structure returned by the **fullstat** subroutine.

Description

The **/usr/include/sys/fullstat.h** file defines the data structure returned by the **fullstat** and **ffullstat** subroutines. This file also defines the *Command* parameters used by the **fullstat** and **ffullstat** subroutines. The **fullstat** structure contains the following fields:

Note: Time is measured in seconds since 00:00:00 GMT, January 1, 1970.

`st_dev`	ID of device containing a directory entry for this file. The file serial number 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 **/usr/include/sys/mode.h** file.
`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** file.
`fst_type`	Type of v-node.
`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.

Implementation Specifics

The following fields are maintained for source-level compatibility with previous versions of the operating system:

```
fst_uid_rev_tag
fst_gid_rev_tag
fst_nid
```

The **fullstat.h** file is part of Base Operating System (BOS) Runtime.

Related Information

The **mode.h** file, **stat.h** file, **statfs.h** file, **types.h** file, **vmount.h** file.

The **statx**, **stat**, **lstat**, **fstax**, **fstat**, **fullstat**, or **ffullstat** subroutine.

The Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

fxconst.inc File

Purpose

Provides **fxfer** function constants for a Pascal file-transfer program.

Description

The **/usr/include/fxconst.inc** file contains the constants used in a programmatical Pascal file-transfer program. Each module that uses the Pascal file-transfer program must include the **fxconst.inc** file. The constants are for use with the 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  */
FXC_CICS        = 4096;           /* ‘1000’x  */
FXC_VSE         = 8192;           /* ‘2000’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 Host Connection Program (HCON).

This file requires the use of the Pascal compiler.

Related Information

The **cfxfer** function, **fxfer** function, and **g32_fxfer** function.

fxfer.h File

Purpose

Contains the **fxc** and **fxs** data structures for the C file-transfer functions.

Description

The **/usr/include/fxfer.h** file defines the C program interface **fxc** structure for the **fxfer** file-transfer function. The *Xfer* parameter of the **fxfer** function specifies a pointer to the **fxc** structure. Each C program module that uses the **fxfer** function must include the **fxfer.h** file. The structures are for use with the C program interface to the HCON file-transfer program.

The C program interface **fxc** structure is defined 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;          /* Option 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
#define FXC_CICS   0x1000
#define FXC_VSE    0x2000
      char *f_logonid;          /* Logon id                       */
      int  loglength;            /* Put here for Pascal stringptr */
      int  f_lrecl;             /* Logical record length            */
      int  f_blksize;            /* Block size                     */
      char *f_inputfld;
    /* Input field for VSE or CICS   */
      int length;                /* Put here for Pascal            */
      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;
      char *f_aix_codepg;   /* Override default AIX codeset name*/
      int codepglength;     /* Put here for Pascal stringptr */
```

```
   } 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     */
};
```

Note: The integer length values are placed within the **/usr/include/fxfer.h** file to allow for the direct conversion of a Pascal **stringptr** to a C program character pointer value. The integer value specifies the actual length of the string as defined in Pascal.

Implementation Specifics

The **fxfer.h** file is part of the Host Connection Program (HCON).

This file requires the use of a C compiler.

Related Information

The **cfxfer** function, **fxfer** function, and **g32_fxfer** function.

fxfer.inc File

Purpose

Contains the **fxc** and **fxs** records for Pascal file-transfer functions.

Description

The **/usr/include/fxfer.inc** file defines the **fxc** record format for the Pascal program interface and is used by the **fxfer** file transfer function. Each Pascal program module that uses the **pfxfer** function must include the **fxfer.inc** file, the **fxconst.inc** file, and the **fxhfile.inc** file. These record formats are for use with the Pascal program interface to the HCON programmatic file transfer.

The **fxconst.inc** file includes the external declarations for the file-transfer Pascal interface routines: **pfxfer** and **pcfxfer**. The **fxhfile.inc** is the Pascal file-transfer invocation file for **pfxfer** and **pcfxfer**. The **fxfer.inc** file contains the `fxs` and `fxc` declarations for the Pascal interface routines.

The **fx_stat***xxxxxx* status file, placed in the **$HOME** directory, contains the status of each file-transfer request made by the application program. 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` and `fxc_opt` 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_inputfld   : stringptr;  /* input mode for VSE or CICS       */
   f_s          : fx_s;       /* S option record                   */
   f_aix_codepg : stringptr;  /* Override default AIX codeset name */
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 Host Connection Program (HCON).

This file requires the use of a Pascal compiler.

Related Information

The **cfxfer** function, **fxfer** function, **g32_fxfer** function.

fxhfile.inc File

Purpose

Contains external declarations for Pascal file transfer.

Description

The **/usr/include/fxhfile.inc** file provides external definitions for the Pascal **pfxfer** and **pcfxfer** file-transfer program functions. The **fxhfile.inc** file 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 Host Connection Program (HCON).

This file requires the use of a Pascal compiler.

Related Information

The **cfxfer** function, **fxfer** function, **g32_fxfer** function.

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            25
#define NUM_LPS             26
/*
 * maximum sessions allowed for single user
 */

#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
```

The **g32_api** structure is:

```
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
 *  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_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 session exists for 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_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 */
                             /* or the specified logical path     */
#define G32_EREAD        -17 /* An error occurred on a ’read’     */
                             /* system call                       */
#define G32_EWRITE       -18 /* An error occurred on a ’write’    */
                             /* system call                       */
#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_EIOCTL       -30 /* The ioctl call to driver failed   */
#define G32_NOTACK       -32 /* The synchronization problem, is   */
                             /* missing g32write function in      */
                             /* the host application              */
#define G32_TIMEOUT      -33 /* Timeout occurred waiting for host */
#define G32_NOATTACH     -34 /* data. API could not allocate or   */
                             /* attach to shared buffers          */
#define G32_OVERRUN      -35 /* Host application overran buffer   */
#define G32_CONN_FAIL    -36 /* Daemon call  connect link failed  */
                             /* Probably means the session name is*/
                             /* already in use                    */
#define G32_ATTN         -37 /* The host application was inter-   */
                             /* rupted with either a SYSREQ or an */
                             /*ATTN key.                          */
                             /* The AIX application should clean  */
                             /*up and exit. */
/*
 *      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
#define G32_FATAL_ERROR -4
#define G32_COMM_REM    -5
/*
 *      constants used in g32_openx
 */
#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
```

Implementation Specifics

The **g32_api.h** file is part of the Host Connection Program (HCON).

This file requires the use of a C compiler.

Related Information

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;
        G32_NOATTACH             = -34;
        G32_OVERRUN              = -35;
        G32_CONN_FAIL            = -36;
        G32_ATTN                 = -37;
/*
 *      Codes returned by g32stat
 */
        G32_NO_ERROR             =  0;
        G32_COMM_CHK             = -1;
        G32_PROG_CHK             = -2;
        G32_MACH_CHK             = -3;
        G32_FATAL_ERROR          = -4;
        G32_COMM_REM             = -5;
```

Implementation Specifics

The **g32const.inc** file is part of the Host Connection Program (HCON).

This file requires the use of a Pascal compiler.

Related Information

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 Host Connection Program (HCON).

This file requires the use of a Pascal compiler.

Related Information

g32_keys.h File

Purpose

Contains common API key value definitions.

Description

The **/usr/include/g32_keys.h** file provides key definitions for use with the HCON C language **g32_send_keys** function. Each module that uses the HCON Pascal **g32_send_keys** function must include the **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 SYSREQ      "\003\033"    /* sys req (SNA only)       */
#define ATTN        "\003\022"    /* attn key (SNA only)      */
#define T_REQ       SYSREQ        /* test/sys req             */
#define HOME        "\002\015"    /* home cursor              */
#define CURSEL      "\002\070"    /* cursor select            */
```

Implementation Specifics

The **g32_keys.h** file is part of the Host Connection Program (HCON).

This file requires the use of a C compiler.

Related Information

g32keys.inc File

Purpose

Contains common API key-value 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 the **g32keys.inc** file.

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 */
SYSREQ         = chr(3) || chr(27);      /* sys request(SNA only) */
ATTN           = chr(3) || chr(18);      /* attN key (SNA only)   */
T_REQ          = chr(3) || chr(27);      /* test/sys request      */
HOME           = chr(2) || chr(13);      /* home cursor           */
CURSEL         = chr(2) || chr(56);      /* cursor select         */
```

Implementation Specifics

The **g32_keys.inc** file is part of the Host Connection Program (HCON).

This file requires the use of a Pascal compiler.

Related Information

The **g32_send_keys** function.

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 the **g32types.inc** file.

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 maximum IO buffer size     */
                         /* characteristic 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_inputfld  : stringptr;   /* Input mode for VSE or CICS     */
   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;
```

Examples

The following example illustrates the use of the Pascal header files:

```
program example(input, output);
        const
                %include /usr/include/g32const.inc
                { user’s constant definitions }
        type
                %include /usr/include/g32types.inc
                { user’s type definitions }
        var
                User_Buffer : packed array[k.1..100]k. of char;
                API_BUF_PTR : integer;
                { user’s variable declarations }
        %include /usr/include/g32hfile.inc
        { user’s external function declarations }
        begin
                API_BUF_PTR = addr(User_Buffer);
                { user’s program }
        end
```

The `API_BUF_PTR` declaration must be an integer and must be assigned the address of the `User_Buffer` declaration.

Implementation Specifics

The **g32types.inc** file is part of the Host Connection Program (HCON).

This file requires the use of a Pascal compiler.

Related Information

Other HCON Pascal header files are **/usr/include/g32const.inc**, **/usr/include/g32hfile.inc**, and **/usr/include/g32keys.inc**.

grp.h File

Purpose

Describes group structure.

Syntax

#include <grp.h>

Description

The **grp.h** header declares the structure group that includes the following members:

```
char    *gr_name  the name of the group
gid_t    gr_gid   numerical group ID
char   **gr_mem   pointer to a null-terminated array of character pointers to member names
```

The **gid_t** type is defined as described in the **sys/types.h** header file.

The following are declared as functions and may also be defined as macros. Function prototypes must be provided for use with an ISO C compiler.

```
struct group  *getgrgid(gid_t);
struct group  *getgrnam(const char *);
int            getgrgid_r(gid_t, struct group *, char *, size_t, struct group **);
int            getgrnam_r(const char *, struct group *, char *, size_t, struct group **);
struct group  *getgrent(void);
void           endgrent(void);
void           setgrent(void);
```

Related Information

The **getgrent**, **endgrent**, **getgrnam**, **getgrgid**, and **getgrgid_r** subroutines.

The **types.h** header file.

iconv.h File

Purpose

Defines types, macros, and subroutines for character code set conversion.

Description

The **/usr/include/iconv.h** file defines types, subroutines, and macros used in character code-set conversion by the iconv family of subroutines and commands. The **iconv.h** file defines the **iconv_t** data type.

Implementation Specifics

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

Related Information

The **genxlt** command, **iconv** command.

The **iconv** subroutine, **iconv_close** subroutine, **iconv_open** subroutine.

National Language Support Overview for Programming in *AIX General Programming Concepts: Writing and Debugging Programs*.

Converters Overview for Programming in *AIX General Programming Concepts: Writing and Debugging Programs*.

List

inode.h File

Purpose

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

Syntax

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

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;
    /*  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_size;

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

/* time last modified  */
struct timestruc_t di_mtime_ts;

/* time last accessed  */
struct timestruc_t di_atime_ts;

/* time last changed inode  */
struct timestruc_t di_ctime_ts;

/*defines for old time_t names */
#define di_mtime   di_mtime_ts.tv_sec
```

```
#define di_atime   di_atime_ts.tv_sec
#define di_ctime   di_ctime_ts.tv_sec

/* 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_info
{
    /* 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;
       }_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 inode if its
      * length is less than D_PRIVATE. Otherwise like
      * 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


    /*
     *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*/
     struct jfsmount *_jfsmnt;   /* ptr to mount data*/
     ushort           _fperpage; /* frag per block */
     ushort           _agsize;   /* frags per ag */
     ushort           _iagsize;  /* inodes per ag */
 }_mt_info;

 /*
  * 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*/
     struct jfsmount *_jfsmnt;   /* ptr to mount data*/
     ushort           _fperpage; /* frag per block */
     ushort           _agsize;   /* frags per ag */
     ushort           _iagsize;  /* inodes per ag */
     ushort           _compress  /* > 0 if data comp */
 }_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
#    define di_jfsmnt   _di_info._mt_info._jfsmnt
#    define di_fperpage _di_info._mt_info._fperpage
#    define di_agsize   _di_info._mt_info._agsize
#    define di_iagsize  _di_info._mt_info._iagsize

     /*
      * 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 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
#  define di_jfsmnt     _di_info._mt_info._jfsmnt
#  define di_fperpage   _di_info._mt_info._fperpage
#  define di_agsize     _di_info._mt_info._agsize
#  define di_iagsize    _di_info._mt_info._iagsize
#  define di_compress   _di_info._mt_info._compress

    /*
     * 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 Base Operating System (BOS) Runtime.

Related Information

The **filsys.h** file, **stat.h** file, **types.h** file.

Files Overview.

Directory Overview and File Systems Overview for

inttypes.h File

Purpose

Contains fixed size integral types.

Syntax

#include <inttypes.h>

Description

The **inttypes.h** header includes definitions of, at least, the following types:

int8_t	8-bit signed integral type.
int16_t	16-bit signed integral type.
int32_t	32-bit signed integral type.
int64_t	64-bit signed integral type.
uint8_t	8-bit unsigned integral type.
uint16_t	16-bit unsigned integral type.
uint32_t	32-bit unsigned integral type.
uint64_t	64-bit unsigned integral type.
intptr_t	Signed integral type large enough to hold any pointer.
uintptr_t	Unsigned integral type large enough to hold any pointer.

Implementation Specifics

Related Information

ipc.h File

Purpose

Describes the structures that are used by the subroutines that perform interprocess communications operations.

Syntax

#include <sys/ipc.h>

Description

The **ipc.h** file defines the following symbolic constants, types, and structures:

Symbolic Constants:

```
IPC_CREAT     create entry if key doesn't exist
IPC_EXCL      fail if key exists
IPC_NOWAIT    error if request must wait
IPC_PRIVATE   private key
IPC_RMID      remove identifier
IPC_SET       set options
IPC_STAT      get options
IPC_ALLO      Centry currently allocated
IPC_R         read or receive permission
IPC_W         write or send permission
IPC_NOERROR   truncates a message if too long
SHM_SIZE      change segment size (shared mem only)
```

The structure **ipc_perm** contains the following members:

```
uid_t                   uid            owner's user id
gid_t                   gid            owner's group id
uid_t                   cuid           creator's user id
gid_t                   cgid           creator's group id
mode_t                  mode           access modes
unsigned short          seq            slot usage sequence number
key_t                   key            key
```

The types **uid_t**, **gid_t**, **mode_t**, and **key_t** are as defined in **<sys/types.h>**.

The following is declared as a function:

```
key_t         ftok(const char *,  int);
```

Implementation Specifics

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

Related Information

The **types.h** file.

The **ftok** subroutine.

iso646.h File

Purpose

Provides alternate spellings.

Syntax

#include <iso646.h>

Description

The **iso646.h** header file defines the following eleven macros (on the left) that expand to the corresponding tokens (on the right):

```
and       &&
and_eq    &=
bitand    &
bitor     |
compl     ~
not       !
not_eq    !=
or        ||
or_eq     |=
xor       ^
xor_eq    ^=
```

limits.h File

Purpose

Defines implementation limits identified by IEEE POSIX 1003.

Description

The **limits.h** 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 follows:

Symbol	Value	Explanation
CHAR_BIT	8	Number of bits in a variable of type char
CHAR_MAX	255	Maximum value of a variable of type char
CHAR_MIN	0	Minimum value of a variable of type char
INT_MAX	2,147,483,647	Maximum value of a variable of type int
INT_MIN	-2,147,483,648	Minimum value of a variable of type int
LONG_MAX	2,147,483,647	Maximum value of a variable of type long
LONG_MIN	-2,147,483,648	Maximum value of a variable of type long
SCHAR_MAX	127	Maximum value of a variable of type signed char
SCHAR_MIN	-128	Minimum value of a variable of type signed char
SHRT_MAX	32,767	Maximum value of a variable of type short
SHRT_MIN	-32,768	Maximum value of a variable of type short
UCHAR_MAX	255	Maximum value of a variable of type unsigned char
UINT_MAX	4,294,967,295	Maximum value of a variable of type unsigned int
ULONG_MAX	4,294,967,295	Maximum value of a variable of type unsigned long
USHRT_MAX	65,535	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	512	No fewer than the number of bytes specified by the **MAX_INPUT** symbol are allowed in a terminal input queue.
NGROUPS_MAX	32	Maximum size of the concurrent group list.
PASS_MAX	32	Maximum number of bytes in a password (not including the null terminator).Only eight characters of password information are significant.
PID_MAX	**INT_MAX**	Maximum value for a processID.
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 that can be attained on a lightly loaded system.

Symbol	Value	Explanation
ARG_MAX	24,576>	Maximum length (in bytes) of arguments for the **exec** subroutine, including the environment

Note: 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
MAX_CANON	256	Maximum number of bytes in a canonical input line
OPEN_MAX	32767	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 null terminator)
PATH_MAX	512	Maximum number of bytes in a path name (not including the null terminator)

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	32,768	Maximum number of bytes guaranteed to be written automatically to a pipe.

Implementation Specifics

This file is provided for POSIX compatibility.

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

Related Information

The **values.h** file.

The **exec** subroutine, **pathconf** or **fpathconf** subroutine.

The Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

math.h File

Purpose

Defines math subroutines and constants.

Description

The **/usr/include/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 macro, which is used as an error-return value:

HUGE_VAL	Specifies the maximum value of a double-precision floating-point number: +infinity on machines that support IEEE-754 and **DBL_MAX** otherwise.

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 __ (double underscore) 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.

If **_XOPEN_SOURCE** variable is defined, the following mathematical constants are defined for your convenience. The values are of type double and are accurate to the precision of this type. That is, the machine value is the mathematical value rounded to double precision.

M_E	Base of natural logarithms (*e*)
M_LOG2E	Base-2 logarithm of *e*
M_LOG10E	Base-10 logarithm of *e*
M_LN2	Natural logarithm of 2
M_LN10	Natural logarithm of 10
M_PI	Pi, the ratio of the circumference of a circle to its diameter
M_PI_2	Value of pi divided by 2
M_PI_4	Value of pi divided by 4
M_1_PI	Value of 1 divided by pi
M_2_PI	Value of 2 divided by pi
M_2_SQRTPI	Value of 2 divided by the positive square root of pi
M_SQRT2	Positive square root of 2
M_SQRT1_2	Positive square root of 1/2

Related Information

The **values.h** file.

Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

mode.h File

Purpose

Defines the interpretation of a file mode.

Description

This version of the 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 by using the **chmod** 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, and 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 supplementary group ID 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 Any file that has the **S_IXACL** bit set 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. The **/usr/include/sys/acl.h** file defines the format of the ACL.

S_ITCB Bit

trusted Any file that has the **S_ITCB** bit set is part of the Trusted Computing Base (TCB). Only files in the TCB can acquire privilege on a trusted path. Only files in the TCB 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

journaled Any file that has the **S_IJRNL** bit set is defined as a journaled file. Updates to a journaled file are added to a log atomically. All directories and system files have the journaled attribute, which 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 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 the file or to search the file's directory.

S_IEXEC Permits the owner of a file to execute the file or to search the file's directory.

S_IRWXG Permits a file's group to read, write, and execute 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 the file or to search the file's directory.

S_IRWXO Permits others to read, write, and execute 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 the file or to search the file's directory.

Implementation Specifics

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

Related Information

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

The **chmod** command, **tsh** command.

The Header Files Overview, defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

msg.h File

Purpose

Describes the structures that are used by the subroutines that perform message queueing operations.

Syntax

#include <sys/msg.h>

Description

The **msg.h** file defines the following symbolic constants, types, and structures:

Types:

```
unsigned int  msgqnum_t;
unsigned int  msglen_t;
```

Symbolic Constants:

MSG_NOERROR	no error if big message */
MSG_R	read permission */
MSG_W	write permission */
MSG_RWAIT	a reader is waiting for a message */
MSG_WWAIT	a writer is waiting to send */
MSG_STAT	Number of bytes to copy for IPC_STAT command
MSGXBUFSIZE	the length of everything but mtext[1] and padding
MSG_SYSSPACE	for rmsgsnd() flags
XMSG	for rmsgrcv() flags

There is one msg queue id data structure for each q in the system. The **msqid_ds** structure contains the following members:

```
struct ipc_perm    msg_perm;           operation permission
struct
void               *__msg_first;       ptr to first message on q
void               *__msg_last;        ptr to last message on q
unsigned int        __msg_cbytes;      current # bytes on q
msgqnum_t          msg_qnum;           # of messages on q
msglen_t           msg_qbytes;         max # of bytes on q
pid_t              msg_lspid;          pid of last msgsnd
pid_t              msg_lrpid;          pid of last msgrcv
```

```
time_t              msg_stime;          last msgsnd time
time_t              msg_rtime;          last msgrcv time
time_t              msg_ctime;          last change time
int                 __msg_rwait;        wait list for message
receive
int                 __msg_wwait;        wait list for message send
unsigned short      __msg_reqevents;    select/poll requested
events
```

The **msg_hdr** struct contains the following members:

```
time_t              mtime;              time message was sent
uid_t               muid;               author's effective uid
gid_t               mgid;               author's effective gid
pid_t               mpid;               author's process id
mtyp_t              mtype;              message type
```

There is one msg structure for each message that may be in the system. The msg structure contains the following members:

```
struct msg          *msg_next;          ptr to next message on q
struct msg_hdr      msg_attr;           message attributes
unsigned int        msg_ts;             message text size
char                *msg_spot;          pointer to message text
```

The structure **msgbuf** is the user message buffer template for **msgsnd** and **msgrcv** system calls and contains the following members:

```
mtyp_t              mtype;              message type
char                mtext[1];           message text
```

The **msgxbuf** structure is the user message buffer template for the **msgxrcv** system call and contains the following members:

```
time_t               mtime;             time message was sent
uid_t                muid;              author's effective uid
gid_t                mgid;              author's effective gid
pid_t                mpid;              author's process id
mtyp_t               mtype;             Message type
char                 mtext[1];          Message text
```

The **msginfo** structure contains the following members:

```
int             msgmax,        max message size
int             msgmnb,        max # bytes on queue
int             msgmni,        # of message queue identifiers
int             msgmnm;        max # messages per queue identifier
```

The **time_t**, **size_t**, **off_t**, **mtyp_t**, **pid_t**, and **gid_t** types are as defined in **<sys/types.h>**.

The following are declared as functions:

```
int msgget(key_t, int);
int msgrcv(int, void *, size_t, long, int);
int msgsnd(int, const void *, size_t, int);
int msgctl(int, int, struct msqid_ds *);
int msgxrcv(int, struct msgxbuf*, int, long, int);
```

In addition, all of the symbols from **<sys/ipc.h>** will be defined when this header is included.

Implementation Specifics

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

Related Information

param.h File

Purpose

Describes system parameters.

Description

Certain parameters vary for different hardware that uses the operating system. These parameters are defined in the **/usr/include/sys/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 **/usr/include/sys/limits.h** file rather than the **param.h** file.

Implementation Specifics

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

Related Information

The **exec** subroutine.

The Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

poll.h File

Purpose

Defines the structures and flags used by the **poll** subroutine.

Description

The **/usr/include/sys/poll.h** 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.

During a polling operation on both file descriptors and message queues, the *ListPointer* parameter points to a **pollist** structure, which can specify either file descriptors or pointers and message queues. The program must define the **pollist** structure in the following form:

```
struct pollist {
   struct pollfd fdlist[f];
   struct pollmsg msglist[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* variable specifies the number of elements in the array.
pollmsg[*m*]	This structure defines an array of message queue identifiers. The *m* variable specifies the number of elements in the array.

A **POLLIST** macro is also defined in the **poll.h** 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** 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.

Related Information

The **fp_poll** kernel service, **fp_select** kernel service, **selnotify** kernel service.

The **poll** subroutine, **select** subroutine.

The Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

The Input and Output Handling Programmer's Overview in *AIX General Programming Concepts: Writing and Debugging Programs* describes the files, commands, and subroutines used for

pthread.h File

Purpose

Lists threads.

Syntax

#include <pthread.h>

Description

The **pthread.h** header defines the following symbols:

PTHREAD_CANCEL_ASYNCHRONOUS
PTHREAD_CANCEL_ENABLE
PTHREAD_CANCEL_DEFERRED
PTHREAD_CANCEL_DISABLE
PTHREAD_CANCELED
PTHREAD_COND_INITIALIZER
PTHREAD_CREATE_DETACHED
PTHREAD_CREATE_JOINABLE
PTHREAD_EXPLICIT_SCHED
PTHREAD_INHERIT_SCHED
PTHREAD_MUTEX_DEFAULT
PTHREAD_MUTEX_ERRORCHECK
PTHREAD_MUTEX_NORMAL
PTHREAD_MUTEX_INITIALIZER
PTHREAD_MUTEX_RECURSIVE
PTHREAD_ONCE_INIT
PTHREAD_PRIO_INHERIT
PTHREAD_PRIO_NONE
PTHREAD_PRIO_PROTECT
PTHREAD_PROCESS_SHARED
PTHREAD_PROCESS_PRIVATE
PTHREAD_RWLOCK_INITIALIZER
PTHREAD_SCOPE_PROCESS
PTHREAD_SCOPE_SYSTEM

The **pthread_attr_t**, **pthread_cond_t**, **pthread_condattr_t**, **pthread_key_t**, **pthread_mutex_t**, **pthread_mutexattr_t**, **pthread_once_t**, **pthread_rwlock_t**, **pthread_rwlockattr_t**, and **pthread_t** types are defined as described in **sys/types.h**.

The following are declared as functions and may also be declared as macros. Function prototypes must be provided for use with an ISO C compiler.

Inclusion of the **pthread.h** header will make visible symbols defined in the headers **sched.h** and **time.h**.

Related Information

The **pthread_attr_init**, **pthread_attr_getguardsize**, **pthread_attr_setscope**, **pthread_cancel**, **pthread_cleanup_push**, **pthread_cond_init**, **pthread_cond_signal**, **pthread_cond_wait**, **pthread_condattr_init**, **pthread_create**, **pthread_detach**, **pthread_equal**, **pthread_exit**, **pthread_getconcurrency**, **pthread_getschedparam**, **pthread_join**, **pthread_key_create**, **pthread_key_delete**, **pthread_mutex_init**, **pthread_mutex_lock**, **pthread_mutex_setprioceiling**, **pthread_mutexattr_init**, **pthread_mutexattr_gettype**, **pthread_mutexattr_setprotocol**, **pthread_once**, **pthread_self**, **pthread_setcancelstate**, **pthread_setspecific**, **pthread_rwlock_init**, **pthread_rwlock_rdlock**, **pthread_rwlock_unlock**, **pthread_rwlock_wrlock**, **pthread_rwlockattr_init** subroutines.

The **sched.h** and **time.h** header files.

pwd.h File

Purpose

Describes password structure.

Syntax

#include <pwd.h>

Description

The **pwd.h** header provides a definition for struct passwd, which includes at least the following members:

```
char       *pw_name      user’s login name
uid_t       pw_uid       numerical user ID
gid_t       pw_gid       numerical group ID
char       *pw_dir       initial working directory
char       *pw_shell     program to use as shell
```

The **gid_t** and **uid_t** types are defined as described in **sys/types.h**.

The following are declared as functions and may also be defined as macros. Function prototypes must be provided for use with an ISO C compiler.

```
struct   passwd    *getpwuid(uid_t);
int                 getpwnam_r(const char *, struct passwd *, char *, size_t, struct passwd **);
int                 getpwuid_r(uid_t, struct passwd *, char *, size_t, struct passwd **);
void                endpwent(void);
struct   passwd    *getpwent(void);
void                setpwent(void);
```

Related Information

The **endpwent**, **getpwnam**, **getpwuid**, and **getpwuid_r** subroutines.

sem.h File

Purpose

Describes the structures that are used by subroutines that perform semaphore operations.

Description

The **/usr/include/sys/sem.h** 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 *SemaphoreID* parameter of the **semop** subroutine. This value can be a positive integer, a negative integer, or 0:

i If the current process has write permission, the positive integer value of this field is added to the value of the `semval` field 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 the `semval` field 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 the value of the `semval` field.

  If the `semval` field 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 **errno** global variable to **EAGAIN**.

  If the value of the `semval` field 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` field associated with the specified semaphore and suspends execution of the calling process until one of the following conditions is met:

  - The value of the `semval` field becomes greater than or equal to the absolute value of the `sem_op` field. When this occurs, the value of the `semncnt` vield associated with the specified semaphore is decremented, the absolute value of the `sem_op` field is subtracted from `semval` value and, if the **SEM_UNDO** flag is set in the `sem_flg` field, the absolute value of the `sem_op` field is added to the *Semadj* value of the calling process for the specified semaphore.
  - The semaphore specified by the *SemaphoreID* parameter for which the calling process is awaiting action is removed from the system (see the **semctl** subroutine). When this occurs, the **errno** global variable 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 the `semncnt` field 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 0 in this field causes one of the following actions:

  - If the `semval` field is 0, the **semop** subroutine returns a value of 0.
  - If the `semval` field is not equal to 0 and the **IPC_NOWAIT** flag is set, the **semop** subroutine returns a value of -1 and sets the **errno** global variable to **EAGAIN**.
  - If `semval` is not equal to 0 and the **IPC_NOWAIT** flag is not set, the **semop** subroutine increments the `semzcnt` field associated with the specified semaphore and suspends execution of the calling process until one of the following conditions is met:
    - The value of the `semval` field becomes 0, at which time the value of the `semzcnt` field associated with the specified semaphore is decremented.
    - The semaphore specified by the *SemaphoreID* parameter for which the calling process is awaiting action is removed from the system. When this occurs, the **errno** global variable 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 the `semzcnt` field 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 0 for an operation, the value 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 value of the `semval` field of the semaphore when the process terminates.

SEM_ORDER Specifies whether to perform atomically or individually the operations specified by the *SemaphoreOperations* array of the **semop** subroutine. (This flag is valid only when included in the *SemaphoreOperations*[0].*sem_flg* parameter, the first operation in the *SemaphoreOperations* array.)

If the **SEM_ORDER** flag 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 the **SEM_ORDER** flag 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, the **semop** subroutine sets the **SEM_ERR** flag 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**.

Implementation Specifics

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

Processes Programmer's Guide.

Related Information

The **atexit** subroutine, **exec** subroutines, **exit** subroutine **fork** subroutine, **semctl** subroutine, **semget** subroutine, **semop** subroutine, **sigaction** subroutine.

The Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

sgtty.h File

Purpose

Provides the terminal interface for the Berkeley line discipline.

Description

The **sgtty.h** file defines the structures 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** file. The **winsize** structure and the ioctl functions that use it are described in tty Subsystem Overview in *AIX General Programming Concepts: Writing and Debugging Programs*.

> **Note:** AIX Version 4 supports the Berkeley line discipline for compatibility with older applications. However, it is strongly recommended to use the POSIX compliant line discipline, which interface is described in the **termios.h** file.

Basic sgtty.h Modes

Basic ioctl functions use the **sgttyb** structure defined in the **sgtty.h** 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. As a result, the line is usually disconnected.
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. The `sg_ospeed` field has the same values as the `sg_ispeed` field.

`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:

ALLDELAY Delays algorithm selection.

BSDELAY Selects backspace delays. Backspace delays are currently ignored. Possible values are BS0 or BS1.

VTDELAY Selects form-feed and vertical-tab delays:

- **FF0** Specifies no delay.
- **FF1** Specifies one delay of approximately 2 seconds.

CRDELAY Selects carriage-return delays:

- **CR0** Specifies no delay.
- **CR1** Specifies one delay. The delay lasts approximately 0.08 seconds.
- **CR2** Specifies one delay. The delay lasts approximately 0.16 seconds.
- **CR3** Specifies one delay. The delay pads lines to be at least 9 characters at 9600 baud.

TBDELAY Selects 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 0.10 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 0.10 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 **EVENP** and **ODDP** flags control both parity checking on input and parity generation on output in COOKED and CBREAK mode (unless the LPASS8 bit is enabled). Even parity is generated on output unless the **ODDP** flag is set and the **EVENP** flag 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 specified by the **LFLUSHO** bit. 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 is 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 Maps 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 input editing. Character and word erase, line kill, input reprint, and special treatment of the backslash character and the EOT character are disabled.

TANDEM Enables automatic flow control (TANDEM mode), which causes the system to produce a stop character (Ctrl-S) when 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 character specified by the `t_stopc` field is accepted on input as the character that stops output and is produced on output as the character to stop input. The character specified by the `t_startc` field is accepted on input as the character that restarts output and is produced on output as the character to restart input.

Basic ioctl Operations

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 Fetches the basic parameters associated with the terminal and stores them in the **sgttyb** structure that is pointed to.

TIOCSETP Sets the parameters according to the **sgttyb** structure that is pointed to. The interface delays until output stops, then throws away any unread characters before changing the modes.

TIOCSETN Has the same effect as the **TIOCSETP** value but does not delay or flush input. Input is not preserved, however, when changing to or from the RAW mode.

For the following codes, the *Value* parameter is ignored:

TIOCEXCL Sets exclusive-use mode; no further opens are permitted until the file is closed.

TIOCNXCL Turns off exclusive-use mode.

TIOCHPCL When the file is closed for the last time, hangs up the terminal. This is useful when the line is associated with a modem used to place outgoing calls.

For the following code, the *Value* parameter is a pointer to an integer.

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 applies to the FREAD and FWRITE bits defined in the **fcntl.h** file. If the FREAD bit is set, all characters waiting in input queues are flushed. If the FWRITE bit is set, all characters waiting in output queues are flushed.

> **Note:** The FREAD and FWRITE bits cannot be used unless the **_KERNEL** flag is set.

In the following codes, the argument is 0 unless specified otherwise:

TIOCSTI	The *Value* parameter points to a character that the system pretends has 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.
TIONREAD	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 bit is set, 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** file, for which the **sgtty.h** 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 that 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 newline 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	Gets the special characters and puts them in the specified structure.
TIOCSETC	Sets 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 Backspaces on erase rather than echoing erase.

LPRTERA Printing terminal erase mode.

LCRTERA Erases character echoes as Backspace-Space-Backspace.

LTILDE Converts ~ (tilde) to ‘ (grave) on output (for terminals that do not recognize the tilde as an ASCII character).

LMDMBUF Stops and starts output when carrier drops.

LLITOUT Suppresses output translations.

LTOSTOP Sends 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 Passes all 8 bits through on input, in any mode.

LCTLECH Echoes input control characters as Ctrl-*X*, delete as Ctrl-?.

LPENDIN Retypes pending input at next read or input character.

LDECCTQ Only Ctrl-Q restarts output after a Ctrl-S.

LNOFLSH Inhibits 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 the character 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 Sets local characters. The argument to this function is a pointer to an **ltchars** structure, which defines the new local special characters.

TIOCGLTC Sets 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 "tty Subsystem Overview" in *AIX General Programming Concepts: Writing and Debugging Programs*.

Implementation Specifics

This file is for Berkeley compatibility.

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

File

/dev/tty The **tty** special file, which is a synonym for the controlling terminal.

Related Information

The **csh** command, **getty** command, **stty** command, **tset** command.

The **ioctl** subroutine, **sigvec** subroutine.

tty Subsystem Overview in *AIX General Programming Concepts: Writing and Debugging Programs*.

shm.h File

Purpose

Describes the structures that are used by the subroutines that perform shared memory operations.

Syntax

#include <sys/shm.h>

Description

The **shm.h** header file defines the following symbolic constants, types, and structures:

Types:

```
typedef unsigned short   shmatt_t;
```

Symbolic Constants:

```
SHMLBA          segment low boundary address multiple
SHMLBA_EXTSHM   SHMLBA value when environment variable EXTSHM=ON
SHM_RDONLY      attach read-only (else read-write)
SHM_RND         round attach address to SHMLBA
SHM_MAP         map a file instead of share a segment
SHM_FMAP        fast file map
SHM_COPY        deferred update
SHM_R           read permission
SHM_W           write permission
SHM_DEST        destroy segment when # attached = 0
ZERO_MEM        for disclaim
SHMHISEG        highest shared memory segment allowed
SHMLOSEG        lowest shared memory segment allowed
NSHMSEGS        number of shared memory segments allowed
```

There is a shared mem id data structure for each shared memory and mapped file segment in the system.

Structures

The structure **shmid_ds** contains the following members:

```
struct ipc_perm    shm_perm      operation permission struct
size_t             shm_segsz     size of segment in bytes
pid_t              shm_lpid      process ID of last shared memory operation
pid_t              shm_cpid      pid of creator
shmatt_t           shm_nattch    number of current attaches
time_t             shm_atime     last shmat time
time_t             shm_dtime     time of last shmdt
time_t             shm_ctime     time of last change by shmctl
```

The structure **shminfo** contains the following members:

```
unsigned int        shmmax          max shared memory segment size
int                 shmmin          min shared memory segment size
int                 shmmni          # of shared memory identifiers
```

The types **pid_t**, **time_t**, **key_t**, and **size_t** are defined as described in **<sys/types.h>**. The following are declared as functions:

```
void  *shmat(int, const void *, int);
int    shmctl(int, int, struct shmid_ds *);
int    shmdt(const void *);
int    shmget(key_t, size_t, int);
```

In addition, all of the symbols from **<sys/ipc.h>** will be defined when this header is included.

Implementation Specifics

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

Related Information

The **types.h** file.

The **shmat**, **shmctl**, **shmdt**, and **shmget** subroutines.

spc.h File

Purpose

Defines external interfaces provided by the System Resource Controller (SRC) subroutines.

Description

The **/usr/include/spc.h** file defines data structures and symbolic constants that are used when calling the SRC subroutines. All subsystems that are controlled by the SRC via sockets or message queues should include this header file.

The **scrreq** data structure in the **spc.h** file defines the format of requests sent to a subsystem by the **srcmstr** daemon. This format is also used by SRC subroutines that send requests to the **srcmstr** daemon.

The **srcreq** data structure contains the following fields:

`mtype`	The message type for the message queue. This field should be included only for message queue subsystems. Programs should be compiled with the **-DSRCBYQUEUE** flag to generate the `mtype` field.
`srchdr`	The SRC header that must be included in all packets sent to and received from an SRC subsystem.
`subreq`	The request to be processed by the SRC subsystem.

The **srchdr** data structure in the **srcreq** data structure contains the return address that is needed to reply to the request. The **srcrrqs** subroutine can be used to extract this information from the request. The **srchdr** data structure is also part of the reply structure returned by a subsystem.

The **srchdr** data structure contains the following fields:

`retaddr`	The return address
`dversion`	The SRC packet version.
`cont`	The continuation indicator. The possible values are:

NEWREQUEST	Used in a request to the **srcmstr** daemon.
CONTINUED	Used in a reply returned by a subsystem, indicating another packet follows.
STATCONTINUED	Used in a status reply returned by a subsystem, indicating another packet follows.
END	Used in a request seen by a subsystem or the last packet in reply sequence.

The **subreq** data structure contains the request to be processed by the subsystem. This same structure is used when calling the **srcsrqt** subroutine to send a request to a subsystem. The **srcsrqt** subroutine formats the required **srchdr** structure. The request is processed by the **srcmstr** daemon and passed on to a subsystem.

The **subreq** data structure contains the following fields:

`object`
: Defines the object on which to act. The possible values are either the **SUBSYSTEM** constant, or a subserver code point. If the object is a subsystem, the value of this field is the **SUBSYSTEM** constant as defined in the **spc.h** file and the `objname` field contains either a null value or the subsystem name. If the object is a subserver, the `object` field value is the code point from the subserver object definition, and the `objname` field is subsystem-defined. The `objname` field can be null, the subserver name, or the subserver process ID. The object value for the subserver cannot equal the value reserved for the subsystem.

`action`
: SRC action to perform. Possible types are:

  START

  STOP

  STATUS or **SRCSTATUS**

  TRACE

  REFRESH

  The values 0-255 are reserved for use by the SRC.

`parm1`
: Modifies the SRC action type by indicating a variable associated with an action. This field is used in a different manner by each of the actions.

`parm2`
: Modifies the SRC action type by indicating a variable associated with an action. This field is used in a different manner by each of the actions.

`objname`
: Name of the object that the request applies to. This can be a subsystem name, a subserver object, or a subserver process ID.

The **srcrep** and **statrep** structures in the **spc.h** file define formats for the replies returned by a subsystem. For more information, see the **srcsrpy** subroutine.

The **srcrep** data structure must be used for replies to start, stop, refresh, and trace requests. It contains the following fields:

`srchdr` Specifies the SRC request/reply (**srchdr**) header.

`svrreply` A reply structure containing the following fields:

`rtncode` Subsystem response to the request. This response is negative on error or subsystem unique message.

`objtype` The object type. This is one of the following:

- **SUBSYSTEM**
- Subserver code point
- Error code

`objtext` Text description.

`objname` Name of the object (subsystem/subserver).

`rtnmsg` Subsystem unique message.

The **statrep** data structure is used for replies to status requests. It contains the following fields:

`srchdr` Specifies the SRC request/reply (**srchdr**) header.

`statcode` A status structure containing the following fields. There may be an array of these structures. This structure contains the following fields:

`objtype` The object type. This is one of the following:

- **SUBSYSTEM**
- Subserver code point
- Error code

`status` Status code. See the **spc.h** file for the symbolic constants that may be used with this field.

`objtext` Text description.

`objname` Name of the object (subsystem/subserver) this reply belongs to.

The **spc.h** file also defines the following constants that are useful in communicating with the **srcmstr** daemon:

SRCNAMESZ The maximum length of an SRC object name (30 bytes, including the null terminator).

SRCPKTMAX The maximum packet size (8192 bytes).

There are also SRC subroutines to manage SRC objects, including subsystems and subservers. The **spc.h** file defines certain symbolic constants which are useful when defining object attributes. The following SRC object descriptors are defined in the **/usr/include/sys/srcobj.h** file:

Respawn action:

RESPAWN=1

ONCE=2

Contact options:

SRCIPC=1

SRCSIGNAL=2

SRCSOCKET=3

Multiple instances of a subsystem are allowed:

SRCYES=1

SRCNO=0

Display subsystem status under certain conditions:

SRCYES=1

SRCNO=0

Default time limit:

TIMELIMIT=20 (seconds)

The **spc.h** file also includes the **/usr/include/srcerrno.h** file, which contains symbolic constants for the errors returned by the SRC library subroutines. The **src_err_msg** subroutine can be used to retrieve the corresponding error message.

SRC Request Structure Example

The following program excerpt is an example of the SRC request (**srcreq**) structure.

```
struct srcreq
{
  long mtype;     /*Contains the message type in the IPC buffer*/
                  /*This field is included if IPC is used and a
                    message queue is expected*/

 struct srchdr srchdr;    /*src header table entry - defined below*/
 struct subreq subreq;    /*the request passed to the subsystem*/
};
```

```
struct srchdr      /*srchdr structure is used by SRC routines*/
                   /*subsystems are not responsible for setting \
                     this*/
{
 struct sockaddr_un retaddr;
 short dversion;   /*the version of the data format*/
 short cont;       /*used to indicate message is continued*/
};

struct subreq
{
 short object;          /*object to act on*/
 short action;          /*action START, STOP, STATUS,TRACE,REFRESH*/
 short parm1;           /* */
 short parm2;           /*               */
 char objname[SRCNAMES];        /*object name*/
};
```

Related Information

The **srcobj.h** file.

The **srcrrqs** subroutine, **srcsrpy** subroutine, **srcsrqt** subroutine, **srcstat** subroutine, **srcstathdr** subroutine, **srcsbuf** subroutine, **srcstattxt** subroutine, **src_err_msg** subroutine.

System Resource Controller (SRC) Overview for Programmers in *AIX General Programming Concepts: Writing and Debugging Programs*.

srcobj.h File

Purpose

Defines object structures used by the System Resource Controller (SRC) subsystem.

Description

The **/usr/include/sys/srcobj.h** header file contains the structures defining SRC objects. The **SRCsubsys** structure contains the following fields:

`subsysname`	String that contains the subsystem name. This string can contain 30 bytes, including the null terminator.
`synonym`	String that contains the subsystem synonym. This string can contain 30 bytes, including the null terminator.
`cmdargs`	String that contains the subsystem command arguments. This string can contain 200 bytes, including the null terminator.
`path`	String that contains the path to the executable files. This string can contain 200 bytes, including the null terminator.
`uid`	User ID for the subsystem.
`auditid`	Audit ID for the subsystem. This value is supplied by the system and cannot be changed by an SRC subroutine.
`standin`	String that contains the path for standard input. This string can contain 200 bytes, including the null terminator.
`standout`	String that contains the path for standard output. This string can contain 200 bytes, including the null terminator.
`standerr`	String that contains the path for standard error. This string can contain 200 bytes, including the null terminator.
`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`	IPC message queue key.
`svrmtype`	IPC message type (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`	String that contains the group name of the subsystem. This string can contain 30 bytes, including the null terminator.

The **SRCsubsvr** structure contains the following fields:

`sub_type`	String that contains the type of the subsystem. This string can contain 30 bytes, including the null terminator.
`subsysname`	String that contains the subsystem name. This string can contain 30 bytes, including the null terminator.
`sub_code`	Subsystem code. This is a decimal number.

The **SRCnotify** structure contains the following fields:

`notifyname`	String that contains the name of the subsystem or group to which the notify method applies. This string can contain 30 bytes, including the null terminator.
`notifymethod`	String that is executed when the SRC detects abnormal termination of the subsystem or group. This string can contain 256 bytes, including the null terminator.

The possible values indicated for the fields are predefined.

Related Information

The **spc.h** file.

The **getssys** subroutine.

Defining Your Subsystem to the SRC in *AIX General Programming Concepts: Writing and Debugging Programs*.

System Resource Controller (SRC) Overview for Programmers in *AIX General Programming Concepts: Writing and Debugging Programs*.

stat.h File

Purpose

Defines the data structures returned by the stat family of subroutines.

Description

The **stat** data structure in the **/usr/include/sys/stat.h** file returns information for the **stat**, **fstat**, **lstat**, **statx**, and **fstatx** subroutines.

The **stat** data structure contains the following fields:

`st_dev`	Device that contains a directory entry for this file.
`st_ino`	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`	File mode. The possible file mode values are given in the description of the **/usr/include/sys/mode.h** file.
`st_nlink`	Number of hard links (alternate directory entries) to the file created using the **link** subroutine.
`st_access`	Field is not implemented. All bits are returned as zero.
`st_size`	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, that mark will be updated to include this page when the file is closed or forced to permanent storage.
`st_rdev`	ID of the device. This field is defined only for block or character special files.
`st_atime`	Time when file data was last accessed.
`st_mtime`	Time when file data was last modified.
`st_ctime`	Time when the file status was last changed.
`st_blksize`	Size, in bytes of each block of the file.
`st_blocks`	Number of blocks actually used by the file (measured in the units specified by the **DEV_BSIZE** constant).
`st_gen`	Generation number of this i-node.

st_type	Type of the v-node for the object. This is one of the following values, which are defined in the **/usr/include/sys/vnode.h** file:

VNON	Unallocated object; this should not occur
VBAD	Unknown type of object
VREG	Regular file
VDIR	Directory file
VBLK	Block device
VCHR	Character device
VLNK	Symbolic link
VSOCK	Socket
VFIFO	FIFO
VMPC	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 **/usr/include/sys/vmount.h** file.
st_flag	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	File owner ID.
st_gid	File group ID.

The **stat64** data structure (AIX versions 4.2 and later) in the **/usr/include/sys/stat.h** file returns information for the **stat64**, **fstat64**, and **lstat64** subroutines. The **stat64** structure contains the same fields as the **stat** structure, with the exception of the following field:

st_size	Number of bytes in a file. The `st_size` field is a 64-bit quantity, allowing file sizes greater than `OFF_MAX`. The `st_size` field of the **stat64** structure is of the type `off64_t`.

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` and `st_mtime` fields 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** subroutines 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.

Implementation Specifics

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

Related Information

The **mode.h** file, **types.h** file, **vmount.h** file.

The **chmod** subroutine, **chownx** subroutine, **link** subroutine, **mknod** or **mkfifo** subroutine, **openx**, **open**, or **creat** subroutine, **pipe** subroutine, **read** subroutine, **shmat** subroutine, **statx**, **stat**, **fstatx**, **fstat**, **fullstat**, or **ffullstat** subroutine, **unlink** subroutine, **utime** subroutine, **write**, **writex**, **writev**, or **writevx** subroutine.

The Header Files Overview defines header files, describes how they are used, and lists several of the header files for which information is provided in this documentation.

statfs.h File

Purpose

Describes the structure of the statistics returned by the **statfs**, **fstatfs**, or **ustat** subroutine.

Description

The **statfs** and **fstatfs** subroutines return information on a mounted (virtual) file system in the form of a **statfs** structure. The **/usr/include/sys/statfs.h** file describes the **statfs** structure, which contains the following fields:

`f_version`	Version number of the **statfs** structure. This value is currently 0.
`f_length`	Length of the buffer that contains the returned information. This value is currently 0.
`f_type`	Type of information returned. This value is currently 0.
`f_bsize`	Optimal block size of the file system.
`f_blocks`	Total number of blocks in the system.
`f_bfree`	Number of free blocks in the file system. The size of a free block is given in the `f_bsize` field.
`f_bavail`	Number of free blocks available to a nonroot user.
`f_files`	Total number of file nodes in the file system.
`f_ffree`	Number of free file nodes in the file system.
`f_fsid`	File system ID.
`f_vfstype`	Type of this virtual file system. Possible values are:

MNT_JFS	AIX Version 3 Journaled File System (JFS)
MNT_NFS	SUN network file system
MNT_CDROM	CD-ROM file system.

`f_fsize`	Fundamental block size of the file system.

f_fname File system name. The value returned by this field depends on the type of file system:

- **JFS** Value returned is copied from the `s_fname` field of the superblock (see the **filsys.h** file format). You can set this value at the time the file system is created by using the **mkfs** command with the **-l** flag. This field gives the preferred mount point for the file system.

  Note: The `s_fname` field in the superblock is only 6 bytes wide. Longer names are truncated to fit.

- **CD-ROM** The field is filled with null bytes because the `f_fname` field is not implemented.
- **NFS** The field is filled with null bytes because the `f_fname` field is not implemented.

f_fpack File system pack name. The value returned by this field depends on the file system type:

- **JFS** The value returned is copied from the `s_fpack` field of the superblock (see the **filsys.h** file format). You can set this value at the time the file system is created using the **mkfs** command with the **-v** flag.

  Note: The `s_fpack` field in the superblock is only 6 bytes wide. Longer pack names are truncated to fit.

- **CD-ROM** The value is copied from the volume identifier field in the primary volume descriptor.
- **NFS** The field is filled with null bytes because the `f_fname` field is not implemented.

f_name_max Maximum length of a component name for this file system.

Note: 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 **/usr/include/ustat.h** file, contains the following fields:

- `f_tfree` Total number of free blocks in the file system. The size of a free block is given in by the **UBSIZE** constant. See the **param.h** file for a description of **UBSIZE**.
- `f_inode` Number of free i-nodes in the file system.
- `f_fname` File system name.
- `f_fpack` File system pack name.

Implementation Specifics

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

Files

statfs.h	Path to the **statfs.h** file.
ustat.h	Path to the **ustat.h** file.

Related Information

The **filsys.h** file format, **param.h** file, **vmount.h** file.

The **statfs**, **fstatfs**, or **ustat** subroutine.

The Header Files Overview defines header files, describes how they are used, and lists several of the documented header files.

statvfs.h File

Purpose

Describes the structure of the statistics that are returned by the **statvfs** subroutines and **fsatvfs** subroutines.

Description

The **statvfs** subroutines and **fsatvfs** subroutines return information on a mounted filesystem in the form of statvfs. The **/usr/include/sys/statvfs.h** file describes the following fields in the **statvfs** subroutine:

`f_bsize`	Preferred file system block size
`f_frsize`	Fundamental file system block size
`f_block`	Total number of block `f_frsize` in the file system.
`f_bfree`	Total number of free blocks of `f_frsize` in the file system.
`f_bavail`	Total number of available blocks of `f_frsize` that can be used by users without root access.
`f_files`	Total number of file nodes in the file system
`f_ffree`	Number of free file nodes in the file system.
`f_favail`	Number of free file nodes that can be user without root access.
`f_fsid`	File system ID.
`f_basetype`	File system type name
`f_flag`	File system flags:

ST_RDONLY	File system is mounted read only
ST_NOSUID	File system does not support set used ID file modes
ST_NODEV	Device opens are not allowed through mounts.

`f_namemax`	Maximum length of a component name for this file system
`f_fstr`	File system specific string.

The following prototypes also appear in the **/usr/include/sys/statvfs.h** file:

```
extern int statvfs(const char *, struct statvfs *);

extern int fsatvfs(int, struct statvfs *);
```

Related Information

systemcfg.h File

Purpose

Defines the **_system_configuration** structure.

Description

The **systemcfg.h** file defines the **_system_configuration** structure. This is a global structure that identifies system characteristics. The **system_configuration** structure is provided in read-only system memory. New fields will be added to the structure in future releases. The attributes in the **_system_configuration** structure have the following values:

architecture
: Identifies the processor architecture. Valid values for AIX Version 4 are:

POWER_RS	Indicates a Power machine.
POWER_PC	Indicates a PowerPC.

implementation
: Identifies the specific version of the processor. Each implementation is assigned a unique bit to allow for efficient checking of implementation sets. The following are examples of valid values (the header file contains more values):

POWER_RS1

POWER_RS2

POWER_RSC

POWER_601

Two special values are also defined: **POWER_RS_ALL** and **POWER_PC_ALL**. These labels are defined as the bit OR of all members of their architecture.

version Identifies the central processing unit (CPU) version number. The following are examples of valid values (the header file contains more values):

PV_RS1	Identifies a Power RS1 machine.
PV_RS2	Identifies a Power RS2 machine.
PV_RS2G	Identifies a Power RS2 machine with graphics assist.
PV_RSC	Identifies a Power RSC machine.
PV_601	Identifies a Power PPC 601 machine.

width Contains the processor data-word size. Valid values are 32 or 64. This value is the maximum data-word size and should not be confused with the current execution mode.

ncpus Identifies the number of CPUs active on a machine. Uniprocessor (UP) machines are identified by a 1. Values greater than 1 indicate multiprocessor (MP) machines.

cache_attr Specifies the cache attributes. Bit 31 determines if the cache is present. If this bit is 1, the cache is present. If bit 31 is 0, then no cache is present and all other cache characteristics are 0. Bit 30 indicates the type of cache. If bit 30 is 1, the cache is combined. Otherwise, if bit 30 is 0 the instruction and data caches are separate.

icache_size Contains the L1 instruction-cache size in bytes. For combined caches, this value is the total cache size.

dcache_size Contains the size of the L1 data-cache size in bytes. For combined caches this the total cache size.

icache_asc Contains the L1 instruction-cache associativity. For a combined cache, this is the combined caches’ associativity.

dcache_asc Contains the L1 data-cache associativity. For a combined cache, this is the combined caches’ associativity.

icache_line Contains the line size in bytes of the L1 instruction cache.

dcache_line Contains the line size in bytes of L1 data cache.

L2_cache_size Contains the size of the L2 cache in bytes. A value of 0 indicates no L2 cache is present.

L2_cache_asc Identifies the associativity of the L2 cache.

tlb_comb Identifies the type of Transaction Lookaside Buffer (TLB) attributes. If the TLB is present, bit 31 is 1. Otherwise, if bit 31 is less than 0, the TLB does not exist and all other TLB characteristics are 0. Bit 30 is 1 if the TLB is combined. If the TLB is separate for the instruction and data cache, bit 30 is 0.

itlb_size Specifies the number of entries in the instruction TLB. For combined TLBs, this is the size of the combined TLB.

dtlb_size Specifies the number of entries in the data TLB. For combined TLBs, this is the size of the combined TLB.

itlb_asc Contains the associativity of the instruction TLB. This attribute's value is equal to the **itlb_size** attribute if the system is fully associative.

dtlb_asc Contains the associativity of the instruction TLB. This attribute's value is equal to the value of the **dtlb_size** attribute if the system is fully associative.

resv_size Contains the PowerPC reservation granule size. This field is a 0 on Power machines.

priv_ick_cnt Contains the number of times lock services attempt to lock a spin lock before blocking AP process/thread in supervisor mode. This a 0 on UP machine. This parameter is used by system-locking services.

prob_lck_cnt Contains the number of times lock services attempt to lock a spin lock before blocking a process or thread in problem state. This a 0 on a UP machine. This parameter is used by system-locking services.

virt_alias Indicates virtual memory aliasing. If 1, the hardware is available for virtual memory aliasing and this ability is used by the system. Virtual memory aliasing is the mapping of one real address to more than one virtual address.

cach_cong Contains the number page index bits that can result in a cache synonym. For machines without cache synonyms, this field is 0.

Implementation Specifics

tar.h File

Purpose

Contains definitions for flags used in the **tar** archive header.

Description

The **/usr/include/tar.h** file contains extended definitions used in the `typeflag` and `mode` fields of the **tar** archive header block. The file also provides values for the required POSIX entries.

tar Archive Header Block

Every file archived using the **tar** command is represented by a header block describing the file, followed by zero or more blocks that give the contents of the file. The end-of-archive indicator consists of two blocks filled with binary zeros. Each block is a fixed size of 512 bytes.

Blocks are grouped for physical I/O operations and groups can be written using a single **write** subroutine operation. On magnetic tape, the result of this write operation is a single tape record. The last record is always a full 512 bytes. Blocks after the end-of-archive zeros contain undefined data.

The header block structure is shown in the following table. All lengths and offsets are in decimal.

Header Block Structure			
Field Name	**Offset**	**Length in Bytes**	**Contents**
`name`	0	100	File name without a / (slash)
`mode`	100	8	File mode
`uid`	108	8	User ID
`gid`	116	8	Group ID
`size`	124	12	Size in bytes
`mtime`	136	12	Latest modification time
`cksum`	148	8	File and header checksum
`typeflag`	156	1	File type
`linkname`	157	100	Linked path name or file name
`magic`	257	6	Format representation for tar
`version`	263	2	Version representation for tar
`uname`	265	32	User name
`gname`	297	32	Group name
`devmajor`	329	8	Major device representation
`devminor`	337	8	Minor device representation
`prefix`	345	155	Path name without trailing slashes

Names are preserved only if the characters are chosen from the POSIX portable file-name character set or if the same extended character set is used between systems. During a read operation, a file can be created only if the original file can be accessed using the **open**, **stat**, **chdir**, **fcntl**, or **opendir** subroutine.

Header Block Fields

Each field within the header block and each character on the archive medium are contiguous. There is no padding between fields. More information about the specific fields and their values follows:

`name` The file's path name is created using this field, or by using this field in connection with the `prefix` field. If the `prefix` field is included, the name of the file is `prefix/name`. This field is null-terminated unless every character is non-null.

`mode` Provides 9 bits for file permissions and 3 bits for SUID, SGID, and SVTX modes. All values for this field are in octal. During a read operation, the designated mode bits are ignored if the user does not have equal (or higher) permissions or if the modes are not supported. Numeric fields are terminated with a space and a null byte. The **tar.h** file contains the following possible values for this field:

Flag	Octal	Description
TSUID	04000	Set user ID on execution.
TSGID	02000	Set group ID on execution.
TSVTX	01000	Reserved.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner.
TUEXEC	00100	Execute or search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute or search by group.
TOREAD	00004	Read by others.
TOWRITE	00002	Write by others.
TOEXEC	00001	Execute or search by other.

`uid` Extracted from the corresponding archive fields unless a user with appropriate privileges restores the file. In that case, the field value is extracted from the password and group files instead. Numeric fields are terminated with a space and a null byte.

`gid` Extracted from the corresponding archive fields unless a user with appropriate privileges restores the file. In that case, the field value is extracted from the password and group files instead. Numeric fields are terminated with a space and a null byte.

`size` Value is 0 when the `typeflag` field is set to **LNKTYPE**. This field is terminated with a space only.

`mtime` Value is obtained from the modification-time field of the **stat** subroutine. This field is terminated with a space only.

`chksum` On calculation, the sum of all bytes in the header structure are treated as spaces. Each unsigned byte is added to an unsigned integer (initialized to 0) with at least 17-bits precision. Numeric fields are terminated with a space and a null byte.

`typeflag` The **tar.h** file contains the following possible values for this field:

Flag	Value	Description
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Reserved.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory. In this case, the `size` field has no meaning.
FIFOTYPE	'6'	FIFO special. Archiving a FIFO file archives its existence, not contents.
CONTTYPE	'7'	Reserved.

If other values are used, the file is extracted as a regular file and a warning issued to the standard error output. Numeric fields are terminated with a space and a null byte.

The **LNKTYPE** flag represents a link to another file, of any type, previously archived. Such linked-to files are identified by each file having the same device and file serial number. The linked-to name is specified in the `linkname` field, including a trailing null byte.

`linkname` Does not use the `prefix` field to produce a path name. If the path name or `linkname` value is too long, an error message is returned and any action on that file or directory is canceled. This field is null-terminated unless every character is non-null.

`magic` Contains the **TMAGIC** value, reflecting the extended **tar** archive format. In this case, the `uname` and `gname` fields will contain the ASCII representation for the file owner and the file group. If a file is restored by a user with the appropriate privileges, the `uid` and `gid` fields are extracted from the password and group files (instead of the corresponding archive fields). This field is null-terminated.

`version` Represents the version of the **tar** command used to archive the file. This field is terminated with a space only.

`uname` Contains the ASCII representation of the file owner. This field is null-terminated.

`gname` Contains the ASCII representation of the file group. This field is null-terminated.

`devmajor` Contains the device major number. Terminated with a space and a null byte.

`devminor` Contains the device minor number. Terminated with a space and a null byte.

`prefix`	If this field is non-null, the file's path name is created using the `prefix/name` values together. Null-terminated unless every character is non-null.

Implementation Specifics

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

Related Information

termio.h File

Purpose

Defines the structure of the **termio** file, which provides the terminal interface for AIX Version 2 compatibility.

Description

The **/usr/include/sys/termio.h** file contains the **termio** structure, which defines special characters as well as the basic input, output, control, and line discipline modes. The **termio.h** file is provided for compatibility with AIX Version 2 applications.

AIX Version 2 applications that include the **termio.h** 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** file contains the following fields:

- `c_iflag`
- `c_oflag`
- `c_cflag`
- `c_lflag`
- `c_cc`

Field Descriptions

`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 1 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 the **IGNBRK** flag is set, a break condition detected on input is ignored, which means that the break condition is not put on the input queue and therefore not read by any process.

BRKINT Interrupts the signal on the break condition. If the **IGNBRK** flag is not set and the **BRKINT** flag 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 the **IGNBRK** nor the **BRKINT** flag is set, a break condition is read as a single \0. If the **PARMRK** flag is set, a break condition is read as \377, \0, \0.

IGNPAR Ignores characters with parity errors. If this flag is set, a byte with a framing or parity error (other than break) is ignored.

PARMRK Marks parity errors. If the **PARMRK** flag is set and the **IGNPAR** flag 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 the **ISTRIP** flag is not set, a valid character of \377 is given to the application as \377, \377. If neither the **IGNPAR** nor the **PARMRK** flag 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 this flag is 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 this flag is set, valid input characters are first stripped to 7 bits; otherwise, all 8 bits are processed.

INLCR Maps a new-line character (NL) to a carriage return (CR) on input. If this flag is set, a received NL character is translated into a CR character.

IGNCR Ignores a CR character. If this flag is set, a received CR character is ignored and not read.

ICRNL Maps a CR character to an NL character on input. If the **ICRNL** flag is set and the **IGNCR** flag is not set, a received CR character is translated into an NL character.

IUCLC Maps uppercase to lowercase on input. If this flag is set, a received uppercase, alphabetic character is translated into the corresponding lowercase character.

IXON Enables start and stop output control. If this flag is set, a received STOP character suspends output and a received START character restarts output. When the **IXON** flag is set, START and STOP characters are not read, but merely perform flow-control functions. When the **IXON** flag is not set, the START and STOP characters are read.

IXANY Enables any character to restart output. If this flag is set, any input character restarts output that was suspended.

IXOFF Enables start-and-stop input control. If this flag is 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 this flag is 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 this flag is 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 this flag is set, the NL character is transmitted as the CR-NL character pair.

OCRNL Maps CR to NL on output. If this flag is set, the CR character is transmitted as the NL character.

ONOCR Indicates no CR output at column 0 (first position). If this flag is set, no CR character is transmitted when at column 0 (first position).

ONLRET NL performs the CR function. If this flag is 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 the **ONLCR**, **OCRNL**, **ONOCR**, nor **ONLRET** flag 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 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 this flag is 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 If this flag is 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 the mask to use before comparing to NL0 and NL1:

- **NL0** Specifies no delay.
- **NL1** Specifies one delay of approximately 0.10 seconds. If the **ONLRET** flag is set, the carriage-return delays are used instead of the new-line delays. If the **OFILL** flag is set, two fill characters are transmitted.

CRDLY Selects the carriage-return delays. This is the 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 the **OFILL** flag is set, two fill characters are transmitted.
- **CR2** Specifies a delay of approximately 0.10 seconds. If the **OFILL** flag 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 the mask to use before comparing to TAB0, TAB1, TAB2, and TAB3. If the **OFILL** flag 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 the **OFILL** flag is set, two fill characters are transmitted.
- **TAB2** Specifies a delay of approximately 0.10 seconds.
- **TAB3** Specifies that tabs are to be expanded into spaces.

BSDLY Selects the backspace delays. This is the mask to use before comparing to BS0 and BS1:

- **BS0** Specifies no delay.
- **BS1** Specifies a delay of approximately 0.05 seconds. If the **OFILL** flag is set, this delay transmits one fill character.

VTDLY Selects the vertical-tab delays. This is the 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 a 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** Specifies a zero baud rate which is used to hang up the connection. If B0 is specified, the 'data terminal ready' signal is not asserted. As a result, the line is usually disconnected. This delay transmits two fill characters. Normally, this disconnects the line.
- **B50** Specifies 50 baud.
- **B75** Specifies 75 baud.
- **B110** Specifies 110 baud.
- **B134** Specifies 134.5 baud.
- **B150** Specifies 150 baud.
- **B200** Specifies 200 baud.
- **B300** Specifies 300 baud.
- **B600** Specifies 600 baud.
- **B1200** Specifies 1200 baud.
- **B1800** Specifies 1800 baud.
- **B2400** Specifies 2400 baud.
- **B4800** Specifies 4800 baud.
- **B9600** Specifies 9600 baud.
- **B19200** Specifies 19,200 baud.
- **B38400** Specifies 38,400 baud.
- **EXTA** Specifies External A.
- **EXTB** Specifies External B.

CSIZE Specifies the character size. These bits specify the character size, in bits, for both transmit and receive operations. The character size does not include the parity bit, if one is used:

- **CS5** 5 bits
- **CS6** 6 bits
- **CS7** 7 bits
- **CS8** 8 bits

CSTOPB Specifies the number of stop bits. If this flag is set, 2 stop bits are sent; otherwise, only 1 stop bit is sent.

CREAD Enables the receiver. If this flag is set, the receiver is enabled. Otherwise, characters are not received.

PARENB Enables parity. If this flag is set, parity generation and detection is enabled and a parity bit is added to each character.

PARODD Specifies odd parity. If parity is enabled, the **PARODD** flag specifies odd parity if set. If parity is enabled and the **PARODD** flag is not set, even parity is used.

HUPCL Hangs up on last close. If this flag is 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 this flag is 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 this flag is 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 this flag is set, it 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 characters. If the **ICANON** flag 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:

- The minimum number of characters specified by the **MIN** value are received.
- The time-out value specified by the **TIME** value has expired since the last character was received.

As a result bursts of input can 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 character values of **MIN** and **TIME** are converted to their ascii equivalents to get the numeric value. The time value represents tenths of seconds.

XCASE Enables canonical uppercase and lowercase presentation. If this flag is set along with the **ICANON** flag, 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 also preceded by a backslash character. In this mode, the output generates and the input accepts the following escape sequences:

<table>
<tr><th>For</th><th>Use</th></tr>
<tr><td>‘ (grave)</td><td>\ ‘</td></tr>
<tr><td>|</td><td>\ !</td></tr>
<tr><td>~</td><td>\ ^</td></tr>
<tr><td>{</td><td>\ (</td></tr>
<tr><td>}</td><td>\ )</td></tr>
<tr><td>\</td><td>\ \</td></tr>
</table>

For example, A is input as `\a`, \n as `\\n`, and \N as `\\\n`.

NOFLSH Disables queue flushing. If this flag is set, the normal flushing of the input and output queues associated with the INTR and QUIT characters is not done.

ECHO Enables echo. If this flag is set, characters are echoed as they are received.

When the **ICANON** flag is set, the following echo functions are possible:

ECHOE Echoes the erase character as Backspace-Space-Backspace. If the **ECHO** and **ECHOE** flags are both set, the ERASE character is echoed as one or more ASCII Backspace-Space-Backspace sequences, which clears the last characters from the screen.

ECHOK Echoes the NL character after kill. If the **ECHOK** flag is set, the NL character is echoed after the kill character is received. This emphasizes that the line is deleted.

ECHONL Echoes the NL character. If the **ECHONL** flag is set, the NL character is echoed even if the **ECHO** flag 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 the **ISIG** flag 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 the **ISIG** flag is set, the INTR character is discarded when processed.

VQUIT Indexes the QUIT special character (Ctrl-\), which is recognized on input if the **ISIG** flag 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 the **ISIG** flag is set, the QUIT character is discarded when processed.

VERASE Indexes the ERASE special character (Backspace), which is recognized on input if the **ICANON** flag 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 the **ICANON** flag is set, the ERASE character is discarded when processed.

VKILL Indexes the KILL special character (Ctrl-u), which is recognized on input if the **ICANON** flag is set. The KILL character deletes the entire line, as delimited by an NL, EOL, EOF, or EOL2 character. If the **ICANON** flag is set, the KILL character is discarded when processed.

VEOF Indexes the EOF special character (Ctrl-d), which is recognized on input if the **ICANON** flag 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 the **ICANON** flag 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 the **ICANON** flag is set. EOL is an additional line delimiter, like NL, and is not normally used.

VEOL2 Indexes the EOL2 special character (Ctrl-@ or ASCII NULL), which is recognized on input if the **ICANON** flag is set. EOL2 is another additional line delimiter, like NL, and 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 "POSIX (termios.h File) Line Discipline" in *AIX General Programming Concepts: Writing and Debugging Programs*.

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 "POSIX (termios.h File) Line Discipline" in *AIX General Programming Concepts: Writing and Debugging Programs*.

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 operations using this form are:

TCGETA Gets the parameters associated with the terminal and stores them in the **termio** structure referenced by the *Structure* parameter.

TCSETA Sets the parameters associated with the terminal from the structure referenced by the *Structure* parameter. The change is immediate.

TCSETAF Waits for the output to drain, and then flushes the input queue and sets the new parameters.

TCSETAW Waits for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

Other ioctl subroutines have the form:

```
ioctl (FileDescriptor, Command, Value)
int Value;
```

The operations using this form are:

Attention: If the user writes an application that performs a **TCSBRK** operation followed by a **TCFLSH** operation prior to closing a port, the last data left in the concentrator box on the 64-port adapter is lost. However, no problem occurs if an SIO, 8-port, or 16-port adapter is used.

TCSBRK
Waits for the output to drain. If the *Value* parameter has a value of 0, it sends a break of 0.25 seconds. A nonzero value causes a break condition of that many milliseconds.

TCSBREAK
Waits for the output to drain. If the *Value* parameter has a value of 0, it sends a break of .25 seconds. A nonzero value causes a break condition of that many milliseconds.

TCXONC
Starts and stops control. If the *Value* parameter has a value of 0, it suspends output. If the *Value* parameter has a value of 1, it restarts suspended output. If the *Value* parameter has a value of 2, it blocks input. If the *Value* parameter has a value of 3, it unblocks input.

TCFLSH
If the *Value* parameter has a value of 0, it flushes the input queue. If the *Value* parameter has a value of 1, it flushes the output queue. If the *Value* parameter has a value of 2, it flushes both the input and output queues.

Another form for ioctl subroutines is:

```
ioctl (FileDescriptor, Command, Structure)
struct csmap* Structure;
```

TCSCSMAP
Sets the code set map from the structure referenced by the structure parameter and rejects any invalid map (any map with 0 length/width or a length greater than **MB_LEN_MAX**). The **/usr/include/sys/tty.h** file contains the structure used for **TCSCSMAP** and **TCGCSMAP** operations.

TCGCSMAP
Returns a copy of the current code set map in the structure referenced by the structure parameter. The **/usr/include/sys/tty.h** file contains the structure used for **TCSCSMAP** and **TCGCSMAP** operations.

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) 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 the SAK sequence on; the **TCSAKOFF** operation turns the SAK sequence off.
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**).

Implementation Specifics

This file is for compatibility with AIX Version 2.

This file is part of 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.

The

termios.h File

Purpose

Defines the structure of the **termios** file, which provides the terminal interface for POSIX compatibility.

Description

The **/usr/include/termios.h** 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 POSIX standard. The **termios.h** file also supports ioctl modem-control operations.

The general terminal interface information is contained in the **termio.h** file. The **termio** structure in the **termio.h** 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** file is implemented. Window and terminal size operations use the **winsize** structure, which is defined in the **ioctl.h** file. The **termios** structure in the **termios.h** file contains the following fields:

- `c_iflag`
- `c_oflag`
- `c_cflag`
- `c_lflag`
- `c_cc`

The **termios.h** file also defines the values for the following parameters of the **tcsetattr** subroutine:

- *OptionalActions*
- *QueueSelector*
- *Action*

The **termios.h** file also supports ioctl modem-control operations.

Field Descriptions

`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 the **IGNBRK** flag 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 the **IGNBRK** flag is not set and the **BRKINT** flag 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 **SIGINT** signal to that foreground process group. If neither the **IGNBRK** nor the **BRKINT** flag is set, a break condition is read as a single \0, or if the **PARMRK** flag is set, as \377, \0, \0.

IGNPAR Ignores characters with parity errors. If this flag is set, a byte with a framing or parity error (other than break) is ignored.

PARMRK Marks parity errors. If the **PARMRK** flag is set, and the **IGNPAR** flag 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 the **ISTRIP** flag is not set, a valid character of \377 is given to the application as \377, \377. If neither the **IGNPAR** nor the **PARMRK** flag 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 this flag is 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 this flag is set, valid input characters are first stripped to 7 bits. Otherwise, all 8 bits are processed.

INLCR Maps a new-line character (NL) to a carriage return (CR) on input. If this flag is set, a received NL character is translated into a CR character.

IGNCR Ignores CR character. If this flag is set, a received CR character is ignored and not read.

ICRNL Maps a CR character to the NL character on input. If the **ICRNL** flag is set and the **IGNCR** flag is not set, a received CR character is translated into a NL character.

IUCLC Maps uppercase to lowercase on input. If this flag is set, a received uppercase, alphabetic character is translated into the corresponding lowercase character.

IXON Enables start and stop output control. If this flag is set, a received STOP character suspends output and a received START character restarts output. When the **IXON** flag is set, START and STOP characters are not read, but merely perform flow-control functions. When the **IXON** flag is not set, the START and STOP characters are read.

IXANY Enables any character to restart output. If this flag is set, any input character restarts output that was suspended.

IXOFF Enables start-and-stop input control. If this flag is 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 input already present in the input stream is not lost. If this flag is not set, no BEL character is echoed; the input in the input queue is discarded if the input stream overflows. This function also requires the **IEXTEN** bit to be set.

`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 this flag is 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 this flag is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This flag is often used in conjunction with the **IUCLC** input mode.

ONLCR Maps NL to CR-NL on output. If this flag is set, the NL character is transmitted as the CR-NL character pair.

OCRNL Maps CR to NL on output. If this flag is set, the CR character is transmitted as the NL character.

ONOCR Indicates no CR output at column 0. If this flag is set, no CR character is transmitted when at column 0 (first position).

ONLRET NL performs CR function. If this flag is 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 the **ONLCR**, **OCRNL**, **ONOCR**, nor **ONLRET** flag is set, the NL character is assumed to do the line-feed function only. The column pointer remains unchanged. The column pointer is set to 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 this flag is 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 If this flag is 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 the mask to use before comparing to NL0 and NL1:

NL0 Specifies no delay.

NL1 Specifies a delay of approximately 0.10 seconds. If the **ONLRET** flag is set, the carriage-return delays are used instead of the new-line delays. If the **OFILL** flag is set, two fill characters are transmitted.

CRDLY Selects the carriage-return delays. This is the 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 the **OFILL** flag is set, this delay transmits two fill characters.

CR2 Specifies a delay of approximately 0.10 seconds. If the **OFILL** flag is set, this delay transmits four fill characters.

CR3 Specifies a delay of approximately 0.15 seconds.

TABDLY Selects the horizontal-tab delays. This is the mask to use before comparing to TAB0, TAB1, TAB2, and TAB3. If the **OFILL** flag 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 the **OFILL** flag is set, two fill characters are transmitted.
- **TAB2** Specifies a delay of approximately 0.10 seconds.
- **TAB3** Specifies that tabs are to be expanded into spaces.

BSDLY Selects the backspace delays. This is the mask to use before comparing to BS0 and BS1:

- **BS0** Specifies no delay.
- **BS1** Specifies a delay of approximately 0.05 seconds. If the **OFILL** flag is set, this delay transmits one fill character.

VTDLY Selects the vertical-tab delays. This is the mask to use before comparing to VT0 and VT1:

- **VT0** Specifies no delay.
- **VT1** Specifies a delay of approximately 2 seconds.

FFDLY Selects the form-feed delays. This is the mask to use before comparing to FF0 and FF1:

- **FF0** Specifies no delay.
- **FF1** Specifies a 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.
  - **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.
  - **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. The character size does not include the parity bit, if one is used:
  - **CS5** 5 bits
  - **CS6** 6 bits
  - **CS7** 7 bits
  - **CS8** 8 bits.
- **CSTOPB** Specifies number of stop bits. If this flag is set, 2 stop bits are sent; otherwise, only 1 stop bit is sent.
- **CREAD** Enables receiver. If this flag is set, the receiver is enabled. Otherwise, characters are not received.
- **PARENB** Enables parity. If this flag is set, parity generation and detection is enabled and a parity bit is added to each character.
- **PARODD** Specifies odd parity. If parity is enabled, the **PARODD** flag specifies odd parity if set. If parity is enabled and the **PARODD** flag is not set, even parity is used.
- **HUPCL** Hangs up on last close. If this flag is 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 this flag is 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 different from 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 this flag is 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** flag is not set, checking is not done.

ICANON — Enables canonical input. If this flag is set, it 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 characters. If the **ICANON** flag 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:

- The minimum number of characters specified by **MIN** are received.
- 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 character values of **MIN** and **TIME** are converted to their ascii equivalents to get the numeric value. The time value represents tenths of seconds.

XCASE — Enables canonical uppercase and lowercase presentation. If this flag is set along with the **ICANON** flag, 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 also preceded by a backslash character. In this mode, the output generates and the input accepts the following escape sequences:

<table>
<tr><th>For</th><th>Use</th></tr>
<tr><td>‘ (grave)</td><td>\ ‘</td></tr>
<tr><td>|</td><td>\ !</td></tr>
<tr><td>~</td><td>\ ^</td></tr>
<tr><td>{</td><td>\ (</td></tr>
<tr><td>}</td><td>\ )</td></tr>
<tr><td>\</td><td>\ \</td></tr>
</table>

For example, A is input as `\a`, \n as `\\n`, and \N as `\\\n`.

NOFLSH — Disables queue flushing. If this flag is 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 pending input. If this flag is set, any input that is pending after a switch from raw to canonical mode is re-input the next time a read operation becomes pending or the next time input arrives. The **PENDIN** flag is an internal-state bit.

IEXTEN — Enables extended (implementation-defined) functions to be recognized from the input data. If this flag is not set, implementation-defined functions are not recognized, and the corresponding input characters are processed as described for the **ICANON**, **ISIG**, **IXON**, and **IXOFF** flags. Recognition of the following special control characters requires the **IEXTEN** flag to be set:

- **VEOL2**
- **VDSUSP**
- **VREPRINT**
- **VDISCRD**
- **VWERSE**
- **VLNEXT**

The functions associated with the following bits also require the **IEXTEN** flag to be set:

- **IMAXBEL**
- **ECHOKE**
- **ECHOPRT**
- **ECHOCTL**

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 this flag is set, characters are echoed as they are received.

When the **ICANON** is set, the following echo functions are also possible:

ECHOE — Echoes the erase character as Backspace-Space-Backspace. If the **ECHO** and **ECHOE** flags are both set and the **ECHOPRT** flag is not set, the ERASE and WERASE characters are echoed as one or more ASCII Backspace-Space-Backspace sequences, which clears the last characters from the screen.

ECHOPRT — If the **ECHO** and **ECHOPRT** flags are both set, the first ERASE and WERASE character in a sequence are echoed 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 nonerase character is echoed. This function also requires the **IEXTEN** bit to be set.

ECHOKE — Backspace-Space-Backspace entire line on line kill. If this flag is set, the kill character is echoed by erasing the entire line from the screen (using the mechanism selected by the **ECHOE** and **ECHOPRT** flags). This function also requires the **IEXTEN** flag to be set.

ECHOK — Echoes the NL character after kill. If the **ECHOK** flag is set and the **ECHOKE** flag is not set, the NL character is echoed after the kill character is received. This emphasizes that the line is deleted.

ECHONL — Echoes the NL character. If the **ECHONL** flag is set, the NL character is echoed even if the **ECHO** flag 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 that results from 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 (preceded by a backslash), the EOF character is not echoed. As a result, because EOT is the default EOF character, terminals that respond to EOT are prevented from hanging up. This function also requires the **IEXTEN** flag to be set.

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 the **ISIG** flag 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 the **ISIG** flag is set, the INTR character is discarded when processed.

VQUIT
Indexes the QUIT special character (Ctrl-\), which is recognized on input if the **ISIG** flag 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 the **ISIG** flag is set, the QUIT character is discarded when processed.

VERASE
Indexes the ERASE special character (Backspace), which is recognized on input if the **ICANON** flag 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 the **ICANON** flag is set, the ERASE character is discarded when processed.

VKILL
Indexes the KILL special character (Ctrl-u), which is recognized on input if the **ICANON** flag is set. The KILL character deletes the entire line, as delimited by a NL, EOL, EOF, or EOL2 character. If the **ICANON** flag is set, the KILL character is discarded when processed.

VEOF
Indexes the EOF special character (Ctrl-d), which is recognized on input if the **ICANON** flag 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 the **ICANON** flag 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 the **ICANON** flag is set. EOL is an additional line delimiter, like NL, and is not normally used.

VEOL2
Indexes the EOL2 special character (Ctrl-@ or ASCII NULL), which is recognized on input if the **ICANON** and **IEXTEN** flags are set. EOL2 is an additional line delimiter, like NL, and is not normally used.

VSTART
Indexes the START special character (Ctrl-q), which is recognized on input if the **IXON** flag is set, and generated on output if the **IXOFF** flag is set. The START character can be used to resume output that has been suspended by a STOP character. If the **IXON** flag 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 the **IXON** flag is set, and generated on output if the **IXOFF** flag is set. The STOP character can be used to with terminals to prevent output from disappearing before it can be read. If the **IXON** flag 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 the **ISIG** flag 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 the **ISIG** flag is set, the SUSP character is discarded when processed.

VDSUSP
Indexes the DSUSP special character (Ctrl-y), which is recognized on input if the **ISIG** and **IEXTEN** flags are set. The DSUSP character generates a **SIGTSTP** signal as the SUSP character 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 the **ISIG** and **IEXTEN** flags are set, the DSUSP character is discarded when processed.

VREPRINT
Indexes the REPRINT special character (Ctrl-r), which is recognized on input if the **ICANON** and **IEXTEN** flags are set. The REPRINT character reprints all characters, preceded by a new line, that have not been read. If the **ICANON** and **IEXTEN** flags are set, the REPRINT character is discarded when processed.

VDISCRD
Indexes the DISCARD special character (Ctrl-o), which is recognized on input if the **ICANON** and **IEXTEN** flags are 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 the **ICANON** and **IEXTEN** flags are set, the DISCARD character is discarded when processed.

VWERSE
Indexes the WERASE special character (Ctrl-w), which is recognized on input if the **ICANON** and **IEXTEN** flags are 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 the **ICANON** and **IEXTEN** flags are set, the WERASE character is discarded when processed.

VLNEXT
Indexes the LNEXT (literal next) special character (Ctrl-v), which is recognized on input if the **ICANON** and **IEXTEN** flags are 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 the **ICANON**, **ECHO**, and **IEXTEN** flags 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 noncanonical mode input processing in "ldterm Line Discipline" in *AIX General Programming Concepts: Writing and Debugging Programs*.

VTIME
Indexes the **TIME** value, which is not a special character. The use of the **TIME** value is described in the discussion of noncanonical mode input processing in "ldterm Line Discipline" in *AIX General Programming Concepts: Writing and Debugging Programs*.

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 by a backslash) so that no special processing is done.

Parameter Value Definitions

The following values for the *OptionalActions* parameter of the **tcsetattr** subroutine are also defined in the **termios.h** 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 until all input received but not read has been discarded before setting the terminal parameters from the **termios** structure.

The following values for the *QueueSelector* 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 data that is received but not read as well as 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** 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** action.

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, used for modem control, are an extension to the POSIX line discipline interface. To use these operations in a program, the program must contain an **#include** statement for the **ioctl.h** 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.

TIOCMGET Gets all modem bits. The argument to this command is a pointer to an integer where the current state of the modem status lines is stored. Which modem status and modem control lines are supported depends on the capabilities of the hardware and the hardware's device driver.

TIOCMSET Sets all modem bits. The argument to this command is a pointer to an integer containing a new set of modem bits. The modem control bits use these bits to turn the modem control lines on or off, depending on whether the bit for that line is set or clear. Any modem status bits are ignored. The actual modem control lines which are supported depend on the capabilities of the hardware and the hardware's device driver.

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.

Implementation Specifics

This file is for POSIX compatibility.

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

Related Information

The **termiox.h** file, **types.h** file.

The **csh** command, **getty** command, **ksh** command, **stty** command, **tset** command.

The **cfgetispeed**, **cfgetospeed**, **cfsetispeed**, **cfsetospeed** subroutine, **ioctl** subroutine, **sigvec** subroutine, **tcdrain** subroutine, **tcflow** subroutine, **tcflush** subroutine, **tcgetattr** subroutine, **tcsendbreak** subroutine, **tcsetattr** subroutine.

termiox.h File

Purpose

Defines the structure of the **termiox** file, which provides the extended terminal interface.

Description

The **termiox.h** file contains an extended terminal interface to support asynchronous hardware flow control. It defines the **termiox** structure and ioctl operations using this structure. The **termiox** structure in the **termiox.h** file contains the following fields:

- `x_hflag`
- `x_cflag`
- `x_rflag`
- `x_sflag`

The **termiox.h** file also supports ioctl hardware flow control operations.

Field	Descriptions

`x_hflag` Describes the hardware flow control mode. The possible modes are:

CDXON Enables CD hardware flow control on output. When set, output will occur only if the 'receive line signal detector' (CD) line is raised by the connected device. If the CD line is dropped by the connected device, output is suspended until the CD line is raised.

CTSXON Enables CTS hardware flow control on output. When set, output will occur only if the 'clear to send' (CTS) line is raised by the connected device. If the CTS line is dropped by the connected device, output is suspended until the CTS line is raised.

DTRXOFF Enables DTR hardware flow control on input. When set, the 'data terminal ready' (DTR) line is raised. If the port needs to have its input stopped, it will drop the DTR line. It is assumed that the connected device will stop its output until DTR is raised.

RTSXOFF Enables RTS hardware flow control on input. When set, the 'request to send' (RTS) line is raised. If the port needs to have its input stopped, it will drop the RTS line. It is assumed that the connected device will stop its output until RTS is raised.

It is not possible to use simultaneously the following flow control modes:

- RTS and DTR
- CTS and CD.

Different hardware flow control modes may be selected by setting the appropriate flags. For example:

- Bi-directional RTS/CTS flow control by setting **RTSXOFF** and **CTSXON**
- Bi-directional DTR/CTS flow control by setting **DTRXOFF** and **CTSXON**
- Modem control or uni-directional **CTS** flow control by setting **CTSXON**.

`x_cflag` Reserved for future use.

`x_rflag` Reserved for future use.

`x_sflag` Describes the open discipline. This field must be set before the first open; it is usually done at configuration time. The possible disciplines are:

DTR_OPEN DTR open discipline. On open, the discipline raises the 'data terminal ready' (DTR) and 'request to send' (RTS) lines, and waits for the 'data carrier detect' (DCD) line to be raised. If the port is opened with the **O_NDELAY** or **O_NONBLOCK** flags, the wait is not done. The DTR and RTS lines are dropped at close time.

WT_OPEN World trade open discipline. On open, the discipline behaves like the DTR open discipline if not in CDSTL mode. In CDSTL mode, the discipline does not raise the DTR line until the 'ring indicate' (RI) line is raised. The DTR line is dropped when the DSR line drops for more than 20 milliseconds.

Hardware Flow Control Operations

The following ioctl operations are used for hardware flow control. To use these operations in a program, the program must contain an **#include** statement for the **ioctl.h** file. The argument to these operations is a pointer to a **termiox** structure.

TCGETX Gets the terminal parameters. The current terminal parameters are stored in the structure.

TCSETX Sets the terminal parameters immediately. The current terminal parameters are set according to the structure. The change is immediate.

TCSETXW Sets the terminal parameters after end of output. The current terminal parameters are set according to the structure. The change occurs after all characters queued for output have been transmitted. This operation should be used when changing parameters will affect output.

TCSETXF Sets the terminal parameters after end of output and flushes input. The current terminal parameters are set according to the structure. All characters queued for output are first transmitted, then all characters queued for input are discarded, and then the change occurs.

Implementation Specifics

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

Related Information

The **termios.h** file.

The **ioctl** subroutine.

tty

types.h File

Purpose

Defines primitive system data types.

Description

The **/usr/include/sys/types.h** file defines data types used in system source code. Since some system data types are accessible to user code, they 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 (`ushort_t`) type, and the `dev_t` type can be 16 bits rather than 32 bits.

Standard Type Definitions

The **types.h** file includes the following standard type definitions, which are defined with a **typedef** statement:

`daddr_t`	Used for disk addresses, except in i-nodes on disk. The **/usr/include/sys/filsys.h** file format describes the format of disk addresses used in i-nodes.
`caddr_t`	Core (memory) address.
`clock_t`	Used for system times as specified in **CLK_TCK**s.
`ino_t`	File system i-node number.
`cnt_t`	File system reference count type.
`dev_t`	Major and minor parts of a device code specify the kind of device and unit number of the device and depend on how the system is customized.
`chan_t`	Channel number (the minor's minor).
`off_t`	File offset, measured in bytes from the beginning of a file or device. `off_t` is normally defined as a signed, 32-bit integer. However, beginning in AIX Version 4.2, in the programming environment which enables large files, `off_t` is defined to be a signed, 64-bit integer.
`offset_t`	64-bit file offset, measured in bytes from the beginning of a file or device.
`off64_t`	64-bit file offset, measured in bytes from the beginning of a file or device. This type definition is valid for AIX Version 4.2 or later.
`soff_t`	32-bit file offset, measured in bytes from the beginning of a file or device. This type definition is valid for AIX Version 4.2 or later.
`paddr_t`	Real address.
`key_t`	IPC key.

`time_t`	Timer ID. Times are encoded in seconds, since 00:00:00 UCT, January 1, 1970.
`nlink_t`	Number of file links.
`mode_t`	File mode.
`uid_t`	User ID.
`gid_t`	Group ID.
`mid_t`	Module ID.
`pid_t`	Process ID.
`slab_t`	Security label.
`mtyp_t`	Interprocess communication (IPC) message type.
`size_t`	Data type is used for sizes of objects.
`ssize_t`	Data type is used for a count of bytes or an error indication.
`uchar_t`	Unsigned char.
`ushort_t`	Unsigned short.
`uint_t`	Unsigned int.
`ulong_t`	Unsigned long.

Unsigned Integers and Addresses

The **types.h** file also includes the following type definitions for unsigned integers and addresses:

```
typedef  struct            _quad { long val[2]; } quad;
typedef  long              swblk_t;
typedef  unsigned 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;
```

Implementation Specifics

This file is part of Includes and Libraries in Base Application Development Toolkit.

Related Information

The **values.h** file.

The **filsys.h** file format.

Header Files Overview.

unistd.h File

Purpose

Defines implementation characteristics identified by POSIX standard.

Description

The **/usr/include/unistd.h** file includes files that contain definitions that are required for compatibility with the POSIX standard:

access.h	Defines symbolic constants for the **access** subroutine.

The **unistd.h** file also defines symbolic constants for the **pathconf**, **fpathconf**, and **sysconf** subroutines. The **unistd.h** file also defines the following symbols, which are used by POSIX applications to determine implementation characteristics:

_POSIX_JOB_CONTROL	POSIX-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** 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.

Implementation Specifics

This file is provided for POSIX compatibility.

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

Related Information

The **limits.h** file, **sys/types.h** file, **termios.h** file, **values.h** file.

The **access** subroutine, **exec** subroutine.

Header Files Overview.

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 **/usr/include/utmp.h** file. The **utmp** structure in this header file contains the following fields:

`ut_user`	User login name.
`ut_line`	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: specifically, the run level of the process.
BOOT_MSG	System boot: specifically, the time of the initial program load (IPL).
OTIME_MSG	Old time: specifically, the time of login.
NTIME_MSG	New time: specifically, the time idle.

`ut_pid`	Process ID.
`ut_type`	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`	Termination status of a process.
`e_exit`	Exit status of a process, marked as the **DEAD_PROCESS** value.
`ut_time`	Time at which the entry was made.

Examples

```
#ifndef -H-UTMP
#define _H_UTMP
#define UTMP_FILE        "/etc/utmp"
#define WTMP_FILE        "/var/adm/wtmp"
#define ILOG_FILE        "/etc/.ilog"
#define ut_name  ut_user

struct utmp
{
   char   ut_user[8];               /* User login name                           */
   char   ut_id[14]                 /* /etc/inittab                              */
   char   ut_line[12];              /* Device(console,lnxx)                      */
   short  ut_type;                  /* Type of entry                             */
   pid_t  ut_pid;                   /* Process ID                                */
   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                       */

     char     ut_host[16];          /* hostname                                  */
 };
                    /*  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                         */

                                     /* waitingforlogin                           */
#define USER_PROCESS       7         /* A user process                            */
#define DEAD_PROCESS       8
#define ACCOUNTING         9
#define UTMAXTYPE ACCOUNTING         /* Largest legal value                        */
                                     /* of ut_type                                 */

   /* Special strings or formats used in the          */
   /* "ut_line" field when accounting for             */
   /* something other than a process.                 */
   /* No string for the ut_line field can be more     */
   /* than 11 chars + a NULL in length.               */

#define RUNLVL_MSG          "run-level %c"
#define BOOT_MSG            "system boot"
#define OTIME_MSG           "old time"
#define TIME_MSG            "new time"

#endif                /* _H_UTMP    */
```

Note: The **who** command extracts information from the **/etc/utmp**, **/var/adm/wtmp**, and **/etc/security/failedlogin** files.

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 in to the system.
/var/adm/wtmp	The path to the **wtmp** file, which contains accounting information about logged-in users.
/etc/security/failedlogin	The path to the **failedlogin** file, which contains a list of invalid login attempts.

Related Information

The **getty** command, **init** command, **login** command, **who** command, **write** command.

The **utmp**, **wtmp**, **failedlogin** file format.

Header Files Overview.

values.h File

Purpose

Defines machine-dependent values.

Description

The **/usr/include/values.h** 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*)	Number of bits in the specified data type
HIBITS	Short integer with only the high-order bit set (0x8000)
HIBITL	Long integer with only the high-order bit set (0x80000000)
HIBITI	Regular integer with only the high-order bit set (same as the **HIBITL** value)
MAXSHORT	Maximum value of a signed short integer (0x7FFF = 32,767)
MAXLONG	Maximum value of a signed long integer (0x7FFFFFFF = 2,147,483,647)
MAXINT	Maximum value of a signed regular integer (same as the **MAXLONG** value)
MAXFLOAT	Maximum value of a single-precision floating-point number
MAXDOUBLE	Maximum value of a double-precision floating-point number
LN_MAXDOUBLE	Natural logarithm of the **MAXDOUBLE** value
MINFLOAT	Minimum positive value of a single-precision floating-point number
MINDOUBLE	Minimum positive value of a double-precision floating-point number
FSIGNIF	Number of significant bits in the mantissa of a single-precision floating-point number
DSIGNIF	Number of significant bits in the mantissa of a double-precision floating-point number
FMAXEXP	Maximum exponent of a single-precision floating-point number
DMAXEXP	Maximum exponent of a double-precision floating-point number
FMINEXP	Minimum exponent of a single-precision floating-point number
DMINEXP	Minimum exponent of a double-precision floating-point number
FMAXPOWTWO	Largest power of two that can be exactly represented as a single-precision floating-point number
DMAXPOWTWO	Largest power of two that can be exactly represented as a double-precision floating-point number

Implementation Specifics

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

Related Information

The **math.h** file, **types.h** file.

Header Files Overview.

vmount.h File

Purpose

Defines the structure of the data associated with a virtual file system.

Description

The **/usr/include/sys/vmount.h** file defines the **vmount** structure. Each active virtual file system (VFS) has a **vmount** structure associated with it. The **vmount** structure contains the mount parameters (such as the mount object and the mounted-over object) for that VFS. The **vmount** data is created when the VFS is mounted. The **mntctl** subroutine returns the VFS 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 VFS 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 VFS. Virtual file systems and their identifiers are deleted at IPL (initial program load).
`vmt_time`	The time at which the VFS 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	AIX Version 3 journaled file system
MNT_NFS	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`	Offset of the data, aligned on a word (32-bit) boundary.
`vmt_size`	Actual size of the data in bytes.
`vmt_data[VMT_OBJECT]`	Name of the device, directory, or file that is mounted.
`vmt_data[VMT_STUB]`	Name of the device, directory, or file that is mounted over.
`vmt_data[VMT_HOST]`	Short (binary) name of the host that owns the mounted object.
`vmt_data[VMT_HOSTNAME]`	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 are specific to the generic file system (GFS) type defined by the `vmt_gfstype` field.
`vmt_data[VMT_ARGS]`	Character-string representation of the arguments supplied when the VFS was created.

Implementation Specifics

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

Related Information

The **mntctl** subroutine, **umount** or **uvmount** subroutine, **vmount or mount** subroutine.

Header Files Overview.

wctype.h File

Purpose

Contains wide-character classification and mapping utilities.

Syntax

#include <wctype.h>

Description

The **wctype.h** header defines the following data types through typedef:

wint_t	As described in **wchar.h**.
wctrans_t	A scalar type that can hold values that represent locale-specific character mappings.
wctype_t	As described in **wchar.h**.

The **wctype.h** header declares the following as functions and may also define them as macros. Function prototypes must be provided for use with an ISO C compiler.

```
int iswalnum(wint_t);int iswalpha(wint_t);int iswcntrl(wint_t);int iswdigit(wint_t);int iswgraph(wint_t);int iswlower(wint_t);int iswprint(wint_t);int iswpunct(wint_t);int iswspace(wint_t);int iswupper(wint_t);int iswxdigit(wint_t);int iswctype(wint_t, wctype_t);wint_t towctrans(wint_t, wctrans_t);wint_t towlower(wint_t);wint_t towupper(wint_t);wctrans_t wctrans(const char *);wctype_t wctype(const char *);
```

The **wctype.h** defines the following macro name:

WEOF	Constant expression of type **wint_t** that is returned by several MSE functions to indicate end-of-file.

For all functions described in this header that accept an argument of type **wint_t**, the value will be representable as a **wchar_t** or will equal the value of **WEOF**. If this argument has any other value, the behaviour is undefined.

The behaviour of these functions is affected by the LC_CTYPE category of the current locale.

Inclusion of the **wctype.h** header may make visible all symbols from the headers **ctype.h**, **stdio.h**, **stdarg.h**, **stdlib.h**, **string.h**, **stddef.h** **time.h** and **wchar.h**.

Related Information

The **iswalnum**, **iswalpha**, **iswcntrl**, **iswdigit**, **iswgraph**, **iswlower**, **iswprint**, **iswpunct**, **iswspace**, **iswupper**, **iswxdigit**, **iswctype**, **setlocale**, **towctrans**, **towlower**, **towupper**, **wctrans**, and **wctype** subroutines.

The **locale.h** and **wchar.h** header files.

x25sdefs.h File for X.25

Purpose

Contains the structures used by the X.25 application programming interface (API).

Description

The **/usr/include/x25sdefs.h** file defines the following structures used by X.25 subroutines.

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 the **x25_ctr_wait** structure.

Structures Used to Establish Calls and Transfer Data

cb_call_struct	Used for calls made and accepted.
cb_data_struct	Used for data 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 permanent virtual circuit (PVC).

Structures Used to Clear, Interrupt and Reset 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 Used to Manage X.25 Communications

cb_circuit_info_struct	Used for information about a virtual circuit.
	Used for information about an X.25 adapter.
cb_link_stats_struct, x25_query_data, and x25_stats	Used for statistics for an X.25 port.

Related Information

cb_call_struct Structure for X.25

Purpose

Used by the **x25_call**, **x25_call_accept**, and **x25_receive** subroutines to pass the X.25 port name, called and calling addresses, facilities, and user data.

Syntax

```
#define X25FLG_D_BIT
 0x00000001
#define X25FLG_LINK_NAME    0x00000002
#define X25FLG_CALLED_ADDR  0x00000004
#define X25FLG_CALLING_ADDR 0x00000008
#define X25FLG_CB_FAC       0x00000010
#define X25FLG_USER_DATA    0x00000020

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;
} ;
```

Flags

X25_FLG_D_BIT	Indicates that the call uses 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.

Fields

`flags`	Notification to the API that the associated field has been used.
`link_name`	Name of the X.25 port used for an incoming call. **Note:** 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 the **cb_fac_struct** structure.
`user_data_len`	Field length for call user data (CUD).
`user_data`	Pointer to call user data (CUD).

Related Information

The x25sdefs.h file.

The **cb_fac_struct** structure, **cb_clear_struct** structure.

cb_circuit_info_struct Structure for X.25

Purpose

Used by the **x25_circuit_query** subroutine to return information about the circuit.

Syntax

```
#define X25FLG_INCOMING_PACKET_SIZE
     0x00000001
#define X25FLG_OUTGOING_PACKET_SIZE
 0x00000002
#define X25FLG_INCOMING_THROUGHPUT_CLASS  0x00000004
#define X25FLG_OUTGOING_THROUGHPUT_CLASS  0x00000008
#define X25FLG_INCOMING_WINDOW_SIZE
 0x00000010
#define X25FLG_OUTGOING_WINDOW_SIZE
 0x00000020

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;
} ;
```

Flags

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

`flags`	Notification to the API that the associated field has been used.
`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.

Related Information

The x25sdefs.h file.

cb_clear_struct Structure for X.25

Purpose

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.

Syntax

```
#define X25FLG_CAUSE        ;0x00000001
#define X25FLG_DIAGNOSTIC   0x00000002
#define X25FLG_CALLED_ADDR  0x00000004
#define X25FLG_CALLING_ADDR 0x00000008
#define X25FLG_CB_FAC       0x00000010
#define X25FLG_USER_DATA    0x00000020

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;
};
```

Flags

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

`flags`	Notification to the API that the associated field has been used.
`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 the **cb_fac_struct** structure.
`user_data_len`	Length of user-data field.
`user_data`	Pointer to user data. This can be used only if "fast select" has been requested in the call-request packet.

Related Information

The x25sdefs.h file.

The **cb_call_struct** structure, **cb_fac_struct** structures.

cb_data_struct Structure for X.25

Purpose

Used by the **x25_send** and **x25_receive** subroutines to pass data control information.

Syntax

```
#define X25FLG_D_BIT 0x00000001
#define X25FLG_Q_BIT 0x00000002
#define X25FLG_M_BIT 0x00000004
#define X25FLG_DATA  0x00000008

struct cb_data_struct
{
  unsigned long flags;
  int data_len;
  unsigned char *data;
} ;
```

Flags

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	Sets the Q-bit in the packet. A nonzero value is converted to a single 1-bit in the packet.
X25FLG_M_BIT	Sets the M-bit in the packet. A nonzero value is converted to a single 1-bit in the packet.
X25_FLG_DATA	Indicates that the `data` field is used.

Fields

`flags`	Notification to the API that the associated field has been used.
`data_len`	Length of data.
`data`	Pointer to actual data.

Related Information

The x25sdefs.h file.

The **cb_call_struct** structure.

cb_dev_info_struct Structure for X.25

Purpose

Used by the **x25_device_query** subroutine to pass device information.

Syntax

```
#define X25FLG_NUA                          0x00000001
#define X25FLG_NO_OF_VCS                    0x00000002
#define X25FLG_MAX_RX_PACKET_SIZE           0x00000004
#define X25FLG_MAX_TX_PACKET_SIZE           0x00000008
#define X25FLG_DEFAULT_SVC_RX_PACKET_SIZE  0x00000010
#define X25FLG_DEFAULT_SVC_TX_PACKET_SIZE  0x00000020

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;
} ;
```

Flags

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

`flags`	Notification to the API that the associated field has been used.
`nua`	Pointer to the network user address (NUA) recorded for the device in ASCIIZ format.
`no_of_vcs`	Number of permanent virtual circuits (PVCs) configured on this device.
`max_rx_packet_size`	Maximum receive packet size in bytes.
`max_tx_packet_size`	Maximum transmit packet size in bytes.
`default_svc_rx_packet_size`	Default receive packet size in bytes.
`default_svc_tx_packet_size`	Default transmit packet size in bytes.

Related Information

The x25sdefs.h file.

Logical Channels and Virtual Circuits.

cb_fac_struct Structure for X.25

Purpose

Used by the **x25_call** and **x25_call_accept** subroutines to pass facilities information.

Syntax

```
#define X25FLG_RPOA
        0x00000001

#define X25FLG_PSIZ
    0x00000002

#define X25FLG_WSIZ
    0x00000004

#define X25FLG_TCLS
    0x00000008

#define X25FLG_REV_CHRG
 0x00000010
#define X25FLG_FASTSEL
  0x000000
20
#define X25FLG_FASTSEL_RSP      0x00000040
#define X25FLG_CUG
     0x00000080

#define X25FLG_OA_CUG           0x0
0000100
#define X25FLG_BI_CUG           0x0
0000200
#define X25FLG_NUI_DATA         0x00000400
#define X25FLG_CI_SEG_CNT       0x00000800
#define X25FLG_CI_MON_UNT       0x00001000
#define X25FLG_CI_CALL_DUR      0x00002000
#define X25FLG_CI_REQUEST       0x00004000
#define X25FLG_CLAMN
   0x00008000

#define X25FLG_CALL_REDR        0x00010000
#define X25FLG_TRAN_DEL         0x00020000
#define X25FLG_CALLING_ADDR_EXT 0x00040000
#define X25FLG_CALLED_ADDR_EXT  0x00080000
#define X25FLG_MIN_TCLS         0x00100000
#define X25FLG_END_TO_END_DEL   0x00200000
#define X25FLG_EXP_DATA         0x00400000
#define X25FLG_FACEXT           0x0
0800000

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];
};
```

Note: The example shows how to code the **cb_fac_struct** structure.

Flags

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 the **X25FLG_FASTSEL** flag 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 as follows, but the lengths associated with individual pointer fields are not explained:

`flags` Notification to the API that the associated field has been used.

`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 the `fac_ext` field are copied directly into the facility field.

When the information is provided by the X.25 network or by the remote data terminal equipment (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 the `fac_ext` field 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 the `psiz_clg` field.

`wsiz_clg` Requested size for the window for packets transmitted by the calling DTE. Values range from 0x01 to 0x07 inclusive.

`wsiz_cld` Requested size for the window for packets to be transmitted by the called DTE. Values 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 = 19,200 bits per second
- 0x0C = 48,000 bits per second

`tcls_cld`	Throughput class request for data sent from the called DTE. Supported values are the same as for the `tcls_clg` field.
`rpoa_id`	Indicates the requested RPOA (Requested Private Operating Agency) 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 also 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. The values for the extended calling and called address flags are:
	`#DEFINE X25_FAC_ADDR_EXT_USE_ENTIRE_OSI_NSAP(0)`
	`#DEFINE X25_FAC_ADDR_EXT_USE_PARTIAL_OSI_NSAP(1)`
	`#DEFINE 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 the `calling_addr_ext` field 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 = 19,200 bits per second
- 0x0C = 48,000 bits per second

`min_tcls_cld`	Throughput class request for data sent from the called DTE. Supported values are the same as for the `min_tcls_clg` field.
`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.

Examples

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";  /* 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;
```

Related Information

The x25sdefs.h file.

The **cb_call_struct** structure, **cb_clear_struct** structure.

cb_int_data_struct Structure for X.25

Purpose

Used by the **x25_interrupt** and **x25_receive** subroutines to pass the interrupt data.

Syntax

```
#define X25FLG_INT_DATA 0x00000001

struct cb_int_struct
{
  unsigned long flags ;
  unsigned char int_data_len;
  unsigned char *int_data;
} ;
```

Flags

X25FLG_INT_DATA	A non-zero value indicates the presence of data in the **cb_int_data** structure.

Fields

`flags`	Notification to the API that the associated field has been used.
`int_data_len`	Length of data in the **cb_int_data** structure.
`int_data`	Interrupt data.

cb_link_name_struct Structure for X.25

Purpose

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.

Syntax

```
#define X25FLG_LINK_NAME 0x00000002

struct cb_link_name_struct
{
  unsigned long flags;
  char *link_name;
};
```

Flags

X25_FLG_LINK_NAME	Indicates that the `link_name` field is used.

Fields

`flags`	Notification to the API that the associated field has been used.
`link_name`	Name of the X.25 port.

Related Information

The x25sdefs.h file.

cb_link_stats_struct, x25_query_data, or x25_stats Structure for X.25

cb_links_stats_struct Structure

Used by the **x25_link_statistics** subroutine to pass statistics about an X.25 port.

```
#define X25FLG_NO_OF_VCS          0x00000008
#define X25FLG_LINK_STATS         0x00000020

struct cb_link_stats_struct
{
  unsigned long flags;
  unsigned int no_of_vcs;
  struct x25_query_data x25_stats;
} ;
```

Flags

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

`flags`	Notification to the API that the associated field has been used
`no_of_vcs`	Number of virtual circuits currently in use for the X.25 port specified
`x25_stats`	Pointer to an **x25_query_data** structure containing CIO and X.25 statistics

x25_query_data Structure

The **x25_query_data** structure is returned from the **CIO_QUERY** ioctl operation. It includes two structures: the **cio_stats** structure containing standard statistics values found in the **sys/comio.h** file, and the **x25_stats** structure containing specific X.25 statistics.

```
struct x25_query_data
{
  struct cio_stats cc;
  struct x25_stats ds;
};
```

x25_stats Structure

The **x25_stats** structure contains specific X.25 statistics.

Note: 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;

  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

`ignored_f_tx`	Number of transmitted frames that have been ignored instead of being transmitted.
`rr_f_tx`	Number of RR (receive ready) frames transmitted.
`rnr_f_tx`	Number of RNR (receive not ready) frames transmitted.
`rej_f_tx`	Number of REJ (reject) frames transmitted.
`info_f_tx`	Number of INFO (information) frames transmitted.
`sabm_f_tx`	Number of SABM (set asynchronous balanced mode) frames transmitted.
`sarm_dm_f_tx`	Number of SARM/DM frames transmitted.
`disc_f_tx`	Number of DISC (disconnect) frames transmitted.
`ua_f_tx`	Number of UA (unnumbered acknowledgment) frames transmitted.
`frmr_f_tx`	Number of FRMR (frame received) frames transmitted.

`bad_nr_f_tx`	Number of frames transmitted with a bad N(R) value.
`unknown_f_tx`	Number of unknown frames transmitted.
`xid_f_tx`	Number of XID frames transmitted.
`bad_length_f_tx`	Number of bad length frames transmitted.
`t1_expirations`	Number of times the T1 timer has timed out.
`lvl2_connects`	Number of times the frame level has been connected.
`lvl2_disconnects`	Number of times the frame level has been disconnected.
`carrier_loss`	Number of times the carrier signal was lost.
`connect_time`	Number of seconds that the link has been connected.
`t4_expirations`	Number of times the T4 timer has timed out.
`t4_n2_expirations`	Number of times the T4 timer has timed out N2 times.
`ignored_f_rx`	Number of received frames that have been ignored instead of being received.
`rr_f_rx`	Number of RR frames received.
`rnr_f_rx`	Number of RNR frames received.
`rej_f_rx`	Number of REJ frames received.
`info_f_rx`	Number of INFO frames received.
`sabm_f_rx`	Number of SABM frames received.
`sarm_dm_f_rx`	Number of SARM/DM frames received.
`disc_f_rx`	Number of DISC frames received.
`ua_f_rx`	Number of UA frames received.
`frmr_f_rx`	Number of FRMR frames received.
`bad_nr_f_rx`	Number of frames received with a bad N(R) value.
`unknown_f_rx`	Number of unknown frames received.
`xid_f_rx`	Number of XID frames received.
`bad_length_f_rx`	Number of bad length frames received.
`data_p_tx`	Number of data packets transmitted.
`rr_p_tx`	Number of RR packets transmitted.
`rnr_p_tx`	Number of RNR packets transmitted.
`interrupt_p_tx`	Number of interrupt packets transmitted.
`interrupt_confirm_p_tx`	Number of interrupt-confirmation packets transmitted.

call-request_p_tx	Number of call-request packets transmitted.
call_accept_p_tx	Number of call-accept packets transmitted.
clear_request_p_tx	Number of clear-request packets transmitted.
clear_confirm_p_tx	Number of clear-confirm packets transmitted.
reset_request_p_tx	Number of reset-request packets transmitted.
reset_confirm_p_tx	Number of reset-confirm packets transmitted.
diagnostic_p_tx	Number of diagnostic packets transmitted.
registration_p_tx	Number of registration packets transmitted.
registration_confirm_p_tx	Number of registration-confirmation packets transmitted.
restart_p_tx	Number of restart packets transmitted.
restart_confirm_p_tx	Number of restart-confirmation packets transmitted.
error_p_tx	Number of error packets transmitted.
t20_expirations	Number of times the T20 timer has timed out.
t21_expirations	Number of times the T21 timer has timed out.
t22_expirations	Number of times the T22 timer has timed out.
t23_expirations	Number of times the T23 timer has timed out.
vc_establishments	Number of times a virtual circuit has been established.
t24_expirations	Number of times the T24 timer has timed out.
t25_expirations	Number of times the T25 timer has timed out.
t26_expirations	Number of times the T26 timer has timed out.
t28_expirations	Number of times the T28 timer has timed out.
data_p_rx	Number of data packets received.
rr_p_rx	Number of RR packets received.
rnr_p_rx	Number of RNR packets received.
interrupt_p_rx	Number of interrupt packets received.
interrupt_confirm_p_rx	Number of interrupt-confirmation packets received.
call-request_p_rx	Number of call-request packets received.
call_accept_p_rx	Number of call-accept packets received.
clear_request_p_rx	Number of clear-request packets received.
clear_confirm_p_rx	Number of clear-confirm packets received.

`reset_request_p_rx`	Number of reset-request packets received.
`reset_confirm_p_rx`	Number of reset-confirm packets received.
`diagnostic_p_rx`	Number of diagnostic packets received.
`registration_p_rx`	Number of registration packets received.
`registration_confirm_p_rx`	Number of registration-confirmation packets received.
`restart_p_rx`	Number of restart packets received.
`restart_confirm_p_rx`	Number of restart-confirmation packets received.
`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. See the `transmit_profile` field for a list of these size values.
`transmit_profile[16]`	A profile of the transmission packet sizes used 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:

Element	**Packet Size**
0	Packet size not known
1	Reserved
2	Reserved
3	Reserved
4	0-15
5	16-31
6	32-63
7	64-127
8	128-255
9	256-511
10	512-1023
11	1024-2047
12	2048-4095
13 -16	Reserved

Related Information

The x25sdefs.h file.

Logical Channels and Virtual Circuits.

cb_msg_struct Structure for X.25

Purpose

Used by the **x25_receive** and **x25_call_clear** subroutines to pass the contents of a received packet to an application.

Syntax

```
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

`msg_type` Type of message being returned, as follows:

X25_CALL_CONNECTED	Call connected: The `cb_call` field points to the **cb_call_struct** structure.
X25_INCOMING_CALL	Incoming call: The `cb_call` field points to the **cb_call_struct** structure.
X25_DATA	Data: The `cb_data` field points to the **cb_data_struct** structure.
X25_DATA_ACK	Data acknowledgment: no buffer.
X25_INTERRUPT	Interrupt: `Theint_data` field 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. The `cb_clear` field points to the **cb_clear_struct** structure. (This should only be received on a **x25_call_clear** call.)
X25_RESET_INDICATION	Reset indication: The `cb_res` field 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 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**.

Related Information

The x25sdefs.h file.

cb_pvc_alloc_struct Structure for X.25

Purpose

Used by the **x25_pvc_alloc** subroutine to pass the name of the X.25 port and the logical channel number.

Syntax

```
#define X25FLG_LINK_NAME 0x00000002
#define X25FLG_LCN       0x00000040

struct cb_pvc_alloc_struct
{
  unsigned long flags;
  char *link_name;
  unsigned int lcn;
} ;
```

Flags

X25_FLG_LCN	Indicates that the `lcn` field is used.
X25_FLG_LINK_NAME	Indicates that the `link_name` field is used.

Fields

`flags`	Notification to the API that the associated field has been used.
`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.

Related Information

The x25sdefs.h file.

Logical Channels and Virtual Circuits.

cb_res_struct Structure for X.25

Purpose

Used by the **x25_reset** and **x25_receive** subroutines to pass the reset cause and diagnostic codes.

Syntax

```
#define X25FLG_CAUSE       0x00000001
#define X25FLG_DIAGNOSTIC  0x00000002

struct cb_res_struct
{
  unsigned long flags;
  unsigned char cause;
  unsigned char diagnostic;
};
```

Flags

X25_FLG_CAUSE	Indicates that the `cause` field is used.
X25_FLG_DIAGNOSTIC	Indicates that the `diagnostic` field is used.

Fields

Structure field definitions are as follows:

Element	Description
`flags`	Notification to the API that the associated field has been used.
`cause`	Cause value of either 0 or in the range 0x80-0xFF, to be inserted in the reset packet.
`diagnostic`	Diagnostic reason to be inserted in the packet. The CCITT default value is 0.

Related Information

The x25sdefs.h file.

ctr_array_struct Structure for X.25

Purpose

Used by the **x25_ctr_wait** subroutine to pass the counter identifier and a value to be exceeded.

Syntax

```
#define X25FLG_CTR_ID     0x00000001
#define X25FLG_CTR_VALUE  0x00000002

struct ctr_array_struct
{
  unsigned long flags;
  int ctr_id;
  int ctr_value;
} ;
```

Flags

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

`flags`	Notification to the API that the associated field has been used.
`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. When the number of messages exceeds the value, the **x25_ctr_wait** subroutine returns control to the calling program.

Related Information

The x25sdefs.h file.

Chapter 5. Directories

Directories contain directory entries. Each entry contains a file or subdirectory name and an i-node (index node reference) number. To increase speed and enhance the use of disk space, the data in a file is stored at various locations throughout the computer's memory. The i-node contains the addresses used to locate all of the scattered blocks of data associated with a file. The i-node also records other information about the file, including time of modification and access, access modes, number of links, file owner, and file type. It is possible to link several names for a file to the same i-node by creating directory entries with the **ln** command.

Because directories often contain information that should not be available to all users of the system, directory access can be protected. See "File Ownership and User Groups" in *AIX Version 4.3 System User's Guide: Operating System and Devices* for more information.

Understanding Types of Directories

Directories can be defined by the system or the system administrator, or you can define your own directories. The system-defined directories contain specific kinds of system files, such as commands. At the top of the file system hierarchy is the system-defined root directory. The root directory is represented by a / (slash) and usually contains the following standard system-related directories:

/bin Symbolic link to the **/usr/bin** directory. In prior UNIX file systems, the **/bin** directory contained user commands that now reside in **/usr/bin** in the new file structure.

/dev Contains device nodes for special files for local devices. The **/dev** directory contains special files for tape drives, printers, disk partitions, and terminals.

/etc Contains configuration files that vary for each machine. Examples include:

- **/etc/hosts**
- **/etc/passwd**

The **/etc** directory contains the files generally used in system administration. Most of the commands that used to reside in the **/etc** directory now reside in the **/usr/sbin** directory. However, for compatibility, it contains symbolic links to the new locations of some executable files. Examples include:

- **/etc/chown** is a symbolic link to the **/usr/bin/chown**.
- **/etc/exportvg** is a symbolic link to the **/usr/sbin/exportvg**.

/export Contains the directories and files on a server that are for remote clients.

/home Serves as a mount point for a file system containing user home directories. The **/home** file system contains per-user files and directories.

In a standalone machine, a separate local file system is mounted over the **/home** directory. In a network, a server might contain user files that should be accessible from several machines. In this case, the server's copy of the **/home** directory is remotely mounted onto a local **/home** file system.

/lib Symbolic link to the **/usr/lib** directory, which contains architecture-independent libraries with names in the form **lib*.a**.

/sbin Contains files needed to boot the machine and mount the **/usr** file system. Most of the commands used during booting come from the boot image's RAM disk file system; therefore, very few commands reside in the **/sbin** directory.

/tmp Serves as a mount point for a file system that contains system-generated temporary files.

/u Symbolic link to the **/home** directory.

/usr Serves as a mount point for a file system containing files that do not change and can be shared by machines (such as executables and ASCII documentation).

Standalone machines mount a separate local file system over the **/usr** directory. Diskless and disk-poor machines mount a directory from a remote server over the **/usr** file system.

/var Serves as a mount point for files that vary on each machine. The **/var** file system is configured as a file system since the files it contains tend to grow. For example, it is a symbolic link to the **/usr/tmp** directory, which contains temporary work files.

Some directories, such as your login or home directory (**$HOME**), are defined and customized by the system administrator. When you log in to the operating system, the login directory is the current directory. If you change directories using the **cd** command without specifying a directory name, the login directory becomes the current directory.

Related Information

Files, Directories, and File Systems for Programmers in *AIX General Programming Concepts: Writing and Debugging Programs* introduces i-nodes, file space allocation, and file, directory, and file system subroutines.

File Systems and Directories Overview in *AIX Version 4.3 System User's Guide: Operating System and Devices* introduces files and directories

/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 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 (BNU) 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 BNU **uucico** daemon, **cu** command, or **tip** command places a call to a remote system, it puts 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 Base Operating System (BOS) Runtime.

Related Information

The **connect** subcommand for the ATE command, **ct** command, **cu** command, **pdelay** command, **pshare** command, **slattach** command, **tip** command.

/usr/lib/hcon Directory

Purpose

Contains files used by the Host Connection Program (HCON).

Description

The **/usr/lib/hcon** directory contains files used by the Host Connection Program (HCON). It contains color and keyboard definition files, terminal definition files, HCON API subdirectories, AUTOLOG example scripts, 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_ktbl	Default binary keyboard-definition table

The color and keyboard definition tables in the **/usr/lib/hcon** directory specify defaults for use by HCON emulator sessions. The **hconutil** command allows users to customize color and keyboard definition tables.

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 LFT, 5081, 3151, 3161, 3162, 3163, and 3164 terminals.
dec.ti.H	Terminal definitions for DEC VT100 and DEC VT220 terminals.
wyse.ti.H	Terminal definition for the WYSE WY-50 and WYSE WY-60 terminals.

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 the operating system windows
h	Binary terminal definition files for color and monochrome LFT
i	Binary terminal definition files for the 5081, 3151, 3161, 3162, and 3163 terminals
j	Binary terminal definition file for use with operating system windows
v	Binary terminal definition files for the DEC VT100 and DEC VT220 terminals
w	Binary terminal definition files for the WYSE WY-50 and WYSE WY-60 terminals

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 interfacing to MVS/TSO host systems, including the **instalapi** program
vm	API programs to use in interfacing to VM/CMS host systems, including the **instalapi** program

AUTOLOG Example Scripts

The **/usr/lib/hcon** directory contains several example files for the AUTOLOG facility. These files are:

File	Contents
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 Host Connection Program (HCON).

Files

/usr/lib/hcon/terminfo directory	Contains terminal definitions.

Related Information

The **hconutil** command creates customized color and keyboard definition tables.

Customizing HCON for System Management in *3270 Host Connection Program 2.1 and 1.3.3 for AIX: Guide and Reference* discusses the options available to you for customizing HCON color and keyboard definitions and terminal definitions.

HCON Programming Examples in *3270 Host Connection Program 2.1 and 1.3.3 for AIX: Guide and Reference* discusses the API directories.

/var/spool/mqueue Directory for Mail

Purpose

Contains the **log** file and temporary files associated with the messages in the mail queue.

Description

The **/var/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

The **/var/spool/mqueue** directory is part of Base Operating System (BOS) Runtime.

Related Information

The **sendmail** command.

The **syslogd** daemon.

/var/spool/uucp Directory for BNU

Purpose

Stores Basic Networking Utilities (BNU) log, administrative, command, data, and execute files in multiple subdirectories.

Description

The **/var/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 **/var/spool/uucp** directory:

.Admin Contains four administrative files, including:

- **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 Records 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.***)
- Data (**D.***)
- Execute (**X.***)
- Temporary (**TM.***)

Implementation Specifics

This directory is part of the Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

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

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

/var/spool/uucp/.Admin Directory for BNU

Purpose

Contains administrative files used by BNU.

Description

The **/var/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

The **uudemon.cleanu** command.

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

/var/spool/uucp/.Corrupt Directory for BNU

Purpose

Contains copies of files that could not be processed.

Description

The **/var/spool/uucp/.Corrupt** directory contains copies of files that could not be processed by the Basic Networking 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

The **uudemon.cleanu** command.

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

/var/spool/uucp/.Log Directories for BNU

Purpose

Contain the BNU program log files.

Description

The **/var/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 **/var/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 according to one of the following formats:

/var/spool/uucp/.Log/*DaemonName*/*SystemName*

OR

/var/spool/uucp/.Log/*CommandName*/*SystemName*

All activities of the **uucp** command are logged in the *SystemName* file in the **/var/spool/uucp/.Log/uucp** directory. All activities of the **uux** command are logged in the *SystemName* file in the **/var/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 **/var/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 **/var/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 **/var/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

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

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

Working with BNU Log Files in *AIX Version 4.3 System Management Guide: Communications and Networks*.

/var/spool/uucp/.Old Directory for BNU

Purpose

Contains the combined BNU program log files.

Description

The **/var/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 **/var/spool/uucp/.Log** directory. The log files are compacted by the **/usr/sbin/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Files

/var/spool/uucp/.Log directory	Contains BNU program log files.

Related Information

The **uucp** command, **uudemon.cleanu** command, **uulog** command, **uux** command**.**

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

Working with BNU Log Files in *AIX Version 4.3 System Management Guide: Communications and Networks*.

Understanding the

/var/spool/uucp/.Status Directory for BNU

Purpose

Contains information about the status of the BNU program contacts with remote systems.

Description

The **/var/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:

- 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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

The **uutry** command, **Uutry** command.

The **uucico** daemon.

The **time** subroutine.

/var/spool/uucp/SystemName Directories for BNU

Purpose

Contain queued requests for file transfers and command executions on remote systems.

Description

The **/var/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 **/etc/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. The files are stored in the *SystemName* directories:

command (C.*)	Contain directions for the **uucico** daemon concerning file transfers.
data (D.*)	Contain data to be sent to remote systems by the **uucico** daemon.
execute (X.*)	Contain instructions for running commands on remote systems.
temporary (TM.*)	Contain data files after their transfer to the remote system until the BNU program can deliver them to their final destinations (usually the **/var/spool/uucppublic** directory).

Implementation Specifics

These directories are part of the Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

The **uucp** command, **uux** command.

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

Understanding the BNU Daemons, Understanding the BNU File and Directory Structure and Using BNU

/var/spool/uucp/.Workspace Directory for BNU

Purpose

Holds temporary files used internally by file transport programs.

Description

The **/var/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

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

/var/spool/uucp/.Xqtdir Directory for BNU

Purpose

Contains temporary files used by the **uuxqt** daemon to execute remote command requests.

Description

The **/var/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 Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

The **uux** command.

The **uuxqt** daemon.

/var/spool/uucppublic Directory for BNU

Purpose

Stores BNU files until they can be delivered.

Description

The **/var/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 on the size of your installation and the number of files sent to the local **/var/spool/uucppublic** directory by users on remote systems, this directory can 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 a specific system.

Implementation Specifics

This directory is part of the Basic Networking Utilities Program (BNU) in BOS Extensions 1.

Related Information

The **uuclean** command, **uucleanup** command, **uucp** command, **uudemon.cleanu** command, **uuto** command, **uux** command.

Installation of BNU is detailed in the *AIX Version 4.1 Installation Guide* or in the *AIX Version 4.2 Installation Guide*.

Vos remarques sur ce document / Technical publication remark form

Titre / Title : Bull AIX 4.3 Files Reference

Nº Reférence / Reference Nº : 86 A2 79AP 06

Daté / Dated : April 2000

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement.
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : ____________________ Date : ________

SOCIETE / COMPANY : ____________________

ADRESSE / ADDRESS : ____________________

____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

Technical Publications Ordering Form
Bon de Commande de Documents Techniques

To order additional publications, please fill up a copy of this form and send it via mail to:
Pour commander des documents techniques, remplissez une copie de ce formulaire et envoyez-la à :

BULL ELECTRONICS ANGERS
CEDOC
ATTN / MME DUMOULIN
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

Managers / Gestionnaires :
Mrs. / Mme : **C. DUMOULIN** +33 (0) 2 41 73 76 65
Mr. / M : **L. CHERUBIN** +33 (0) 2 41 73 63 96

FAX : +33 (0) 2 41 73 60 19
E–Mail / Courrier Electronique : srv.Cedoc@franp.bull.fr

Or visit our web site at: / Ou visitez notre site web à:
`http://www–frec.bull.com` (PUBLICATIONS, Technical Literature, Ordering Form)

CEDOC Reference # Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
__ __ ____ _ [__]		__ __ ____ _ [__]		__ __ ____ _ [__]	
[_ _] : **no revision number means latest revision** / pas de numéro de révision signifie révision la plus récente					

NOM / NAME : ______________________________ Date : __________

SOCIETE / COMPANY : ______________________________

ADRESSE / ADDRESS : ______________________________

____

PHONE / TELEPHONE : ____________________ FAX : ____________________

E–MAIL : ______________________________

For Bull Subsidiaries / Pour les Filiales Bull :
Identification: ______________________________

For Bull Affiliated Customers / Pour les Clients Affiliés Bull :
Customer Code / Code Client : ______________________________

For Bull Internal Customers / Pour les Clients Internes Bull :
Budgetary Section / Section Budgétaire : ______________________________

For Others / Pour les Autres :
Please ask your Bull representative. / Merci de demander à votre contact Bull.

PLACE BAR CODE IN LOWER LEFT CORNER

BULL ELECTRONICS ANGERS
CEDOC
34 Rue du Nid de Pie – BP 428
49004 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 79AP 06

Utiliser les marques de découpe pour obtenir les étiquettes.
Use the cut marks to get the labels.

AIX

AIX 4.3 Files
Reference

86 A2 79AP 06

AIX

AIX 4.3 Files
Reference

86 A2 79AP 06

AIX

AIX 4.3 Files
Reference

86 A2 79AP 06