NAME

cgetent, **cgetset**, **cgetmatch**, **cgetcap**, **cgetnum**, **cgetstr**, **cgetustr**, **cgetfirst**, **cgetnext**, **cgetclose** – capability database access routines

SYNOPSIS

#include <stdlib.h>

int
cgetent(*char **buf*, *char **db_array*, *char *name*);

int
cgetset(*char *ent*);

int
cgetmatch(*char *buf*, *char *name*);

*char **
cgetcap(*char *buf*, *char *cap*, *char type*);

int
cgetnum(*char *buf*, *char *cap*, *long *num*);

int
cgetstr(*char *buf*, *char *cap*, *char **str*);

int
cgetustr(*char *buf*, *char *cap*, *char **str*);

int
cgetfirst(*char **buf*, *char **db_array*);

int
cgetnext(*char **buf*, *char **db_array*);

int
cgetclose(*void*);

DESCRIPTION

Cgetent( ) extracts the capability rec *name* from the database specified by the NULL terminated file array *db_array* and returns a pointer to a malloc'd copy of it in *buf*. **Cgetent** will first look for files ending in **.db** (see cap_mkdb(1)) before accessing the ASCII file. *Buf* must be retained through all subsequent calls to **cgetmatch**(), **cgetcap**(), **cgetnum**(), **cgetstr**(), and **cgetustr**(), but may then be free'd. On success 0 is returned, 1 if the returned record contains an unresolved **tc** expansion, −1 if the requested record couldn't be found, −2 if a system error was encountered (couldn't open/read a file, etc.) also setting *errno*, and −3 if a potential reference loop is detected (see **tc=** comments below).

Cgetset enables the addition of a character buffer containing a single capability record entry to the capability database. Conceptually, the entry is added as the first "file" in the database, and is therefore searched first on the call to **cgetent**. The entry is passed in *ent*. If *ent* is NULL, the current entry is removed from the database. **Cgetset** must precede the database traversal. It must be called before the **cgetent** call. If a sequential access is being performed (see below), it must be called before the first sequential access call (**cgetfirst** or **cgetnext**), or be directly preceded by a **cgetclose** call. On success 0 is returned and −1 on failure.

Cgetmatch will return 0 if *name* is one of the names of the capability record *buf*, −1 if not.

Cgetcap searches the capability record *buf* for the capability *cap* with type *type*. A *type* is specified using any single character. If a colon (':') is used, an untyped capability will be searched for (see below for explanation of types). A pointer to the value of *cap* in *buf* is returned on success, NULL if the requested capability couldn't be found. The end of the capability value is signaled by a ':' or ASCII NUL (see below for capability database syntax).

Cgetnum retrieves the value of the numeric capability *cap* from the capability record pointed to by *buf*. The numeric value is returned in the *long* pointed to by *num*. 0 is returned on success, −1 if the requested numeric capability couldn't be found.

Cgetstr retrieves the value of the string capability *cap* from the capability record pointed to by *buf*. A pointer to a decoded, NUL terminated, malloc'd copy of the string is returned in the *char ** pointed to by *str*. The number of characters in the decoded string not including the trailing NUL is returned on success, −1 if the requested string capability couldn't be found, −2 if a system error was encountered (storage allocation failure).

Cgetustr is identical to **cgetstr** except that it does not expand special characters, but rather returns each character of the capability string literally.

Cgetfirst, **cgetnext**, comprise a function group that provides for sequential access of the NULL pointer terminated array of file names, *db_array*. **Cgetfirst** returns the first record in the database and resets the access to the first record. **Cgetnext** returns the next record in the database with respect to the record returned by the previous **cgetfirst** or **cgetnext** call. If there is no such previous call, the first record in the database is returned. Each record is returned in a malloc'd copy pointed to by *buf*. **Tc** expansion is done (see **tc=** comments below). Upon completion of the database 0 is returned, 1 is returned upon successful return of record with possibly more remaining (we haven't reached the end of the database yet), 2 is returned if the record contains an unresolved **tc** expansion, −1 is returned if an system error occurred, and −2 is returned if a potential reference loop is detected (see **tc=** comments below). Upon completion of database (0 return) the database is closed.

Cgetclose closes the sequential access and frees any memory and file descriptors being used. Note that it does not erase the buffer pushed by a call to **cgetset**.

CAPABILITY DATABASE SYNTAX

Capability databases are normally ASCII and may be edited with standard text editors. Blank lines and lines beginning with a '#' are comments and are ignored. Lines ending with a '\' indicate that the next line is a continuation of the current line; the '\' and following newline are ignored. Long lines are usually continued onto several physical lines by ending each line except the last with a '\'.

Capability databases consist of a series of records, one per logical line. Each record contains a variable number of ':'-separated fields (capabilities). Empty fields consisting entirely of white space characters (spaces and tabs) are ignored.

The first capability of each record specifies its names, separated by '|' characters. These names are used to reference records in the database. By convention, the last name is usually a comment and is not intended as a lookup tag. For example, the *vt100* record from the **termcap** database begins:

```
d0|vt100|vt100-am|vt100am|dec vt100:
```

giving four names that can be used to access the record.

The remaining non-empty capabilities describe a set of (name, value) bindings, consisting of a names optionally followed by a typed values:

name	typeless [boolean] capability *name* is present [true]
name*T*value	capability (*name*, *T*) has value *value*
name@	no capability *name* exists
name*T*@	capability (*name*, *T*) does not exist

Names consist of one or more characters. Names may contain any character except ':', but it's usually best to restrict them to the printable characters and avoid use of graphics like '#', '=', '%', '@', etc. Types are single characters used to separate capability names from their associated typed values. Types may be any character except a ':'. Typically, graphics like '#', '=', '%', etc. are used. Values may be any number of characters and may contain any character except ':'.

CAPABILITY DATABASE SEMANTICS

Capability records describe a set of (name, value) bindings. Names may have multiple values bound to them. Different values for a name are distinguished by their *types*. **Cgetcap** will return a pointer to a value of a name given the capability name and the type of the value.

The types '#' and '=' are conventionally used to denote numeric and string typed values, but no restriction on those types is enforced. The functions **cgetnum** and **cgetstr** can be used to implement the traditional syntax and semantics of '#' and '='. Typeless capabilities are typically used to denote boolean objects with presence or absence indicating truth and false values respectively. This interpretation is conveniently represented by:

```
(getcap(buf, name, ':') != NULL)
```

A special capability, **tc= name**, is used to indicate that the record specified by *name* should be substituted for the **tc** capability. **Tc** capabilities may interpolate records which also contain **tc** capabilities and more than one **tc** capability may be used in a record. A **tc** expansion scope (i.e., where the argument is searched for) contains the file in which the **tc** is declared and all subsequent files in the file array.

When a database is searched for a capability record, the first matching record in the search is returned. When a record is scanned for a capability, the first matching capability is returned; the capability **:nameT@:** will hide any following definition of a value of type *T* for *name*; and the capability **:name@:** will prevent any following values of *name* from being seen.

These features combined with **tc** capabilities can be used to generate variations of other databases and records by either adding new capabilities, overriding definitions with new definitions, or hiding following definitions via '@' capabilities.

EXAMPLES

```
example|an example of binding multiple values to names:\
        :foo%bar:foo^blah:foo@:\
        :abc%xyz:abc^frap:abc$@:\
        :tc=more:
```

The capability foo has two values bound to it (bar of type '%' and blah of type '^') and any other value bindings are hidden. The capability abc also has two values bound but only a value of type '$' is prevented from being defined in the capability record more.

```
file1:
        new|new_record|a modification of "old":\
                :fript=bar:who-cares@:tc=old:blah:tc=extensions:
file2:
        old|old_record|an old database record:\
                :fript=foo:who-cares:glork#200:
```

The records are extracted by calling **cgetent** with file1 preceding file2. In the capability record new in file1, fript=bar overrides the definition of fript=foo interpolated from the capability record old in file2, who-cares@ prevents the definition of any who-cares definitions in old from being seen, glork#200 is inherited from old, and blah and anything defined by the record extensions is added to those definitions in old. Note that the position of the fript=bar and who-cares@ definitions before tc=old is important here. If they were after, the definitions in old would take precedence.

CGETNUM AND CGETSTR SYNTAX AND SEMANTICS

Two types are predefined by **cgetnum** and **cgetstr**:

name#*number*	numeric capability *name* has value *number*
name=*string*	string capability *name* has value *string*
name#@	the numeric capability *name* does not exist
name=@	the string capability *name* does not exist

Numeric capability values may be given in one of three numeric bases. If the number starts with either '`0x`' or '`0X`' it is interpreted as a hexadecimal number (both upper and lower case a-f may be used to denote the extended hexadecimal digits). Otherwise, if the number starts with a '`0`' it is interpreted as an octal number. Otherwise the number is interpreted as a decimal number.

String capability values may contain any character. Non-printable `ASCII` codes, new lines, and colons may be conveniently represented by the use of escape sequences:

^X	('*X*' & 037)	control-*X*
\b, \B	(ASCII 010)	backspace
\t, \T	(ASCII 011)	tab
\n, \N	(ASCII 012)	line feed (newline)
\f, \F	(ASCII 014)	form feed
\r, \R	(ASCII 015)	carriage return
\e, \E	(ASCII 027)	escape
\c, \C	(:)	colon
\\	(\)	back slash
\^	(^)	caret
nnn	(ASCII octal *nnn*)	

A '\' may be followed by up to three octal digits directly specifies the numeric code for a character. The use of ASCII `NUL`s, while easily encoded, causes all sorts of problems and must be used with care since `NUL`s are typically used to denote the end of strings; many applications use '\200' to represent a `NUL`.

DIAGNOSTICS

Cgetent, **cgetset**, **cgetmatch**, **cgetnum**, **cgetstr**, **cgetustr**, **cgetfirst**, and **cgetnext** return a value greater than or equal to 0 on success and a value less than 0 on failure. **Cgetcap** returns a character pointer on success and a `NULL` on failure.

Cgetent, and **cgetseq** may fail and set *errno* for any of the errors specified for the library functions: `fopen`(2), `fclose`(2), `open`(2), and `close`(2).

Cgetent, **cgetset**, **cgetstr**, and **cgetustr** may fail and set *errno* as follows:

[`ENOMEM`] No memory to allocate.

SEE ALSO

`cap_mkdb`(1), `malloc`(3)

BUGS

Colons (':') can't be used in names, types, or values.

There are no checks for **tc=name** loops in **cgetent**.

The buffer added to the database by a call to **cgetset** is not unique to the database but is rather prepended to any database used.