900 Series HP 3000 Computer Systems

MPE/iX Developer's Kit Reference Manual Volume 2

HEWLETT PACKARD

HP Part No. 36430-90002
Printed in U.S.A. 1994

Second Edition
E0494

FINAL TRIM SIZE : 7.0 in x 8.5 in

FINAL TRIM SIZE : 7.0 in x 8.5 in

FINAL TRIM SIZE : 7.0 in x 8.5 in

Printing History

The following table lists the printings of this document, together with the respective release dates for each edition. The software version indicates the version of the software product at the time that this document was issued. Many product releases do not require changes to the document; therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	**Date**	**Software Version**
First Edition	October 1992	A.00.00
Second Edition	April 1994	C.50.00

Preface

MPE/iX, Multiprogramming Executive with Integrated POSIX, is the latest in a series of operating systems for the HP 3000 line of computers.

In HP documentation and in talking with HP 3000 users, you will encounter references to MPE XL, the direct predecessor of MPE/iX. MPE/iX is a superset of MPE XL. All programs written for MPE XL will run without change under MPE/iX. You can continue to use MPE XL system documentation, although it may not refer to features added to the operating system to support POSIX (for example, hierarchical directories).

Finally, you may encounter references to MPE V, which is the operating system for HP 3000s not based on PA-RISC architecture. MPE V software can be run on the PA-RISC (Series 900) HP 3000s in what is known as *compatibility mode*.

The *MPE/iX Developer's Kit Reference Manual Volume 2* (36430-90002) describes the POSIX/iX library provided with the MPE/iX Developer's Kit (36430A) on 900 Series HP 3000 computer systems. This manual is intended for experienced C programmers.

This manual is organized as follows:

Chapter 1	**Introduction** provides a summary overview of the supplemental libraries and facilities that are available through the MPE/iX Developer's Kit (product number 36430A).
Chapter 2	**SVID IPC Library Function Descriptions** provides information on a set of C/iX library functions that provides interprocess communication services to applications requiring information exchange and resource synchronization between multiple processes.
Chapter 3	**TERMINFO Database** provides information on the database that describes terminal and printer capabilities.
Chapter 4	**CURSES** presents the syntax and descriptions of the CURSES routines and macros, arranged alphabetically.

Conventions

`nonitalics` Within syntax descriptions, nonitalicized words represent literals. Enter them exactly as shown. This includes angle brackets appearing within syntactic descriptions. For example,

```
#include <unistd.h>
```

Nonitalicized words and punctuation characters appear in `computer font`. In the following example, you must provide the keyword, function name, parentheses, and trailing semicolon:

```
int ccode();
```

italics Within syntax descriptions, italicized words denote argument names, program names, or strings that you must replace with an appropriate value. In the following example, you must replace *number* and *denom* with the respective integers you want to pass to the `div` function:

`div(`*number*`, `*denom*`);`

[ ] Within syntax descriptions, italicized brackets surround optional elements. For example, the *item* list in the `scanf()` function call is optional:

`scanf(`*format [, item [, item]...]* `);`

... Within syntax descriptions, a horizontal ellipses indicates that a previous element can be repeated. For example:

[, item]...

Within examples, vertical and horizontal ellipses may show where portions of the example were omitted.

Contents

Tables

1

Introduction

This chapter provides a summary overview of supplemental libraries and facilities that are available through the MPE/iX Developer's Kit (product # 36430A). The POSIX/iX library, also available through the MPE/iX Developer's Kit, is fully described in the *MPE/iX Developer's Kit Reference Manual Volume 1* (36430-90001).

The following topics are discussed in this chapter:

- What is the SVID IPC library?
- What is the TERMINFO database?
- What is Curses?
- How to use this manual.
- Developing applications using the MPE/iX Shell and Utilities.
- Understanding MPE/iX.

What Is the SVID IPC Library?

The SVID IPC library contains a set of C/iX library functions that provides interprocess communication services to applications requiring information exchange and resource synchronization between multiple processes.

These C/iX library functions emulate the behavior of a set of interprocess communication and synchronization functions defined by the *AT&T System V Interface Definition* (SVID2). For this reason, the set of IPC functions provided in the MPE/iX Developer's Kit is referred to as "SVID IPC."

Refer to Chapter 2 for a complete description of the functions available in the SVID IPC library.

What Is the TERMINFO Database?

The `TERMINFO` database describes terminal and printer capabilities. A wide range of capabilities can be defined that include, for example, the number of lines and columns for the device, whether or not the terminal wraps at the right margin, or what character sequence causes a carriage return. The database is used by screen-oriented programs such as VI or `CURSES` programs. By using `TERMINFO` to handle the capabilities of individual devices, a program can work with a variety of devices without any changes to the code.

Refer to Chapter 3 for a complete description of the `TERMINFO` database.

What Is the CURSES Library?

The `CURSES` library contains a set of C/iX library functions that provides screen management services consisting of routines and macros for creating and modifying input and output to a terminal screen. `CURSES` contains routines for creating windows, highlighting text, writing to the screen, reading from user input, and moving the cursor.

`CURSES` is designed to optimize screen update activities. For example, when updating the screen, `CURSES` minimizes the number of characters sent to the terminal to move and update the screen.

`CURSES` is a terminal-independent package, providing a common user interface to a variety of terminal types. Its portability is facilitated by the `TERMINFO` database which contains a compiled definition of each terminal type. By referring to the database information, `CURSES` gains access to low-level details about individual terminals.

Refer to Chapter 4 for a complete description of the routines available in the `CURSES` package.

How to Use This Manual

This manual is intended to be used with the following four manuals:

- *MPE/iX Developer's Kit Reference Manual Volume 1* (36430-90001)
- *HP C/iX Library Reference Manual* (30026-90001)
- *MPE/iX Shell and Utilities Reference Manual, Volumes 1 and 2* (36431-60001)
- *The POSIX.1 Standard - A Programmer's Guide* (36430-90003)

The four manuals listed above contain descriptions of C library functions available through the POSIX/iX library provided with the MPE/iX Developer's Kit. Refer to the *MPE/iX Developer's Kit Reference Manual Volume 1* (36430-90001) for more information about using these manuals.

The POSIX/iX library is an implementation on 900 Series HP 3000 computer systems of many of the C library functions and features defined in

- IEEE Standard 1003.1-1990 (ISO/IEC 9945-1:1990)
- Appendix B of the IEEE P1003.2/D11.2

Note All references to POSIX.1 in this manual refer to the 1990 revision of the POSIX.1 standard, 1003.1-1990.

Developing Applications Using the MPE/iX Shell and Utilities

Application development using libraries provided with the MPE/iX Developer's Kit must be accomplished through the MPE/iX Shell and Utilities, a command interpreter that provides a set of commands and utilities useful for application development. The MPE/iX Shell is based on the Korn Shell, a command interpretor available on many computer systems.

To invoke the MPE/iX Shell from the MPE/iX Command Interpretor (CI), enter either of the following at the CI prompt:

```
:RUN SH.HPBIN.SYS;INFO="-L"
```

```
:SH.HPBIN.SYS -L
```

Note The `L` must be entered in uppercase.

For more information about the MPE/iX Shell and Utilities, refer to the following manuals:

- *MPE/iX Shell and Utilities Reference Manual, Volumes 1 and 2* (36431-60001)
- *MPE/iX Shell and Utilities User's Guide* (36431-90002)

Compiling and linking an application that requires libraries available through the MPE/iX Developer's Kit must be accomplished through the `c89` command available in the MPE/iX Shell. For detailed information about using the `c89` command, refer to the *MPE/iX Shell and Utilities Reference Manual, Volumes 1 and 2* (36431-60001).

Understanding MPE/iX

The MPE/iX Developer's Kit provides facilities that allow you to develop portable applications while minimizing the need to understand underlying MPE/iX operating system features. Some of the topics discussed in the manual require that you have an understanding of underlying features of the MPE/iX operating system.

Additional MPE/iX documentation is available that contains information about MPE/iX features not discussed in detail in this manual. This manual briefly summarizes these features and provides pointers to the manuals where you can acquire additional information.

The following manual provides an introduction to many of the MPE/iX features that you will need to understand:

- *New Features of MPE/iX: Using the Hierarchical File System* (32650-90351)

2

SVID IPC Library Function Descriptions

This chapter describes a set of C/iX library functions that provides interprocess communication services to applications requiring information exchange and resource synchronization between multiple processes.

These C/iX library functions emulate the behavior of a set of interprocess communication and synchronization functions defined by the *AT&T System V Interface Definition* (SVID2). For this reason, the set of IPC functions provided in the MPE/iX Developer's Kit is referred to as "SVID IPC."

This chapter in organized in the following manner:

- Overview of the SVID IPC facilities
- Using the SVID IPC library
- Managing SVID IPC services
- Conformance and implementation considerations
- Detailed descriptions of SVID IPC functions
- Descriptions of SVID IPC headers

Overview of SVID IPC

Developing an application consisting of several processes usually requires some form of information exchange and resource synchronization. Many applications developed for use on UNIX®-based computer systems use the IPC services defined by the *AT&T System V Interface Definition (SVID2).*

The MPE/iX SVID IPC library is provided (along with the POSIX.1/iX library) to enable application developers to substantially reduce the costs associated with successfully porting a UNIX-based application to a Series 900 HP 3000 computer system.

The SVID IPC library provides three facilities and an access control feature:

- Message queues
- Shared memory
- Semaphores
- Access control

For each of the three SVID IPC facilities there are C/iX library functions available for allocation, deallocation, and control of the facilities.

Use an *xxx*`get()` function to either create a resource of the desired type (for example, a shared memory area) or obtain an identifier for an existing resource.

Use an *xxx*`ctl()` function to perform a variety of control operations on an IPC resource. Common control operations include returning usage statistics about a resource and modifying attributes of the resource.

Additional functions are provided in each facility to perform activities unique to the facility. For example, the message queue facility provides functions to write to and read from a specified message queue.

Message queues

Message queues are one form of process-to-process communication. Data sent from a process to a message queue is copied into a data buffer which is then linked into the list of data buffers for that queue.

Processes can read and write to message queues in any arbitrary order. For example, a process can send a message to a queue even if no other process is currently waiting to read a message from the same queue.

Following are the functions available to manage SVID IPC message queues:

`msgctl()`	Perform control operations on a message queue
`msgget()`	Locate or allocate a message queue
`msgrcv()`	Receive a message from a message queue
`msgsnd()`	Send a message to a message queue

Shared memory

The shared memory facility allows processes to communicate via a common memory area mapped to the address space of each process sharing the memory. When a process modifies the contents of a shared memory area, the data is immediately available to other processes.

The most common usage of this facility is to allocate application global data needed by all processes sharing the memory area. An example of shared memory access is when one process writes data to a shared memory area and a second process reads the data from the same shared memory area. In this case, there is no overhead involved in copying the data from process A to process B. Process B has immediate access to the data through the shared memory area.

Following are the functions available to manage SVID IPC shared memory.

`shmat()`	Attach a process to a shared memory area
`shmctl()`	Perform control operations on a shared memory area
`shmdt()`	Detach a process from a shared memory area
`shmget()`	Locate or allocate a shared memory area

Semaphores

The semaphore facility allows processes to synchronize operations when sharing a common resource. An example of semaphore facility use is when two or more processes synchronize access to a commonly shared memory area. In this case, a process performing a write operation to a shared memory area can suspend itself until other processes have completed read operations on the same shared memory area.

Following are the functions available to manage SVID IPC semaphores:

`semctl()`	Perform control functions on the semaphore set.
`semget()`	Locate or allocate a semaphore set.
`semop()`	Operate on the semaphore set.

Access control

Access control is provided through a key value, known to the application using SVID IPC functions, that is associated with each resource being shared by multiple processes. This key is used by different processes who have agreed to manage access to the resource using the shared key. All SVID IPC facilities require the user to supply a key to be used by the `msgget()`, `semget()`, and `shmget()` functions to obtain identifiers.

The `ftok()` function provides a user with a key value that can be used by processes sharing the same resource. The key value returned by `ftok()` is based upon a file name to which the application has access.

There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If a standard is not adhered to, it is possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the *id* parameter of `ftok()`, if used for key generation, should in some sense refer to a project so that keys do not conflict across a given system.

Note The successful management of SVID IPC facilities is predicated upon the assumption that all accessors of a shared resource have agreed to access the resource through a shared key. If a process does not use an agreed-upon scheme to access the shared resource, proper management of that resource cannot be guaranteed.

Using the SVID IPC Library

The SVID IPC library is implemented as a C/iX relocatable library located in the file `/lib/libsvipc.a`.

In order to add the SVID IPC library to your list of relocatable libraries at link time, specify `svipc` with the `-l` option of the `c89` command available through the MPE/iX Shell. For example:

```
c89 -o foo foo.c -lsvipc
```

Managing SVID IPC Services

Interactive utilities are provided through the MPE/iX Command Interpretor to manage the interprocess communication services provided by SVID IPC functions. These utilities are implemented as MPE/iX command files and located in `HPBIN.SYS`. Each utility provides a built-in help facility that contains detailed information on use.

IPCS utility

The `IPCS` utility enables a user with system manager (SM) capability to display status information on SVID IPC services currently allocated on the system. This utility can also be used to display current configuration values.

The `IPCS` utility is invoked by entering:

```
IPCS.HPBIN.SYS
```

IPCRM utility

The `IPCRM` utility enables a user with system manager (SM) capability to interactively remove specific SVID IPC resources from the system, performing the same function as `IPC_RMID` on a *XXX*`ctl()` function call.

The `IPCRM` utility is invoked by entering:

```
IPCRM.HPBIN.SYS
```

SVIPC utility

The `SVIPC` utility enables a user with system manager (SM) capability to interactively modify current SVID IPC configuration limits.

The `SVIPC` utility is invoked by entering:

```
SVIPC.HPBIN.SYS
```

Conformance and Implementation Considerations

The MPE/iX SVID IPC library was implemented to emulate the behavior of the equivalent SVID-conformant IPC functionality available on the Series 800 HP 9000 computer system.

Please read the "Implementation Considerations" section of each function description prior to using the function in order to understand any implementation-defined behavior or behavior that does not conform to the definition of the function in the *AT&T System V Interface Definition*.

SVID IPC Library Function Descriptions

The following section describes SVID IPC library functions in detail. Function descriptions are arranged alphabetically.

ftok

Returns a key used in calls to `msgget()`, `semget()`, and `shmget()` function calls.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok (char *pathname, char id);
```

Parameters

pathname Passes a pointer to a string containing the pathname of a file accessible to the calling process.

id Passes a character used to further qualify the returned key.

Return Values

$\neq$-1 Key value returned.

-1 Error. The specified pathname does not exist or is not accessible to the calling process.

Description

The `ftok()` function can be used to return a key based on both *pathname* and *id*. This key can be used in subsequent calls to the SVID IPC `msgget()`, `semget()`, and `shmget()` functions.

A different key is returned when called with the same pathname but a different character is passed in *id*.

Implementation Considerations

None.

Errors

If an error occurs, `errno` is set to the following value.

`EINVAL`	CAUSE	The *pathname* parameter points to an invalid address, or *pathname* specifies a nonexistent file, or the caller does not have the correct access to the pathname.
	ACTION	Make sure that *pathname* points to a valid and existing pathname to which the caller has access.

See Also

`msgget()`, `semget()`, `shmget()`

msgctl

Provides message control operations.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (int msqid, int cmd, struct msqid_ds *buffer);
```

Parameters

msqid Passes a message queue identifier returned by a call to `msgget()`.

cmd Passes a command defining the control operation to perform. Valid commands are defined in the "Description" section below.

buffer Passes a pointer to a buffer of type `struct msqid_ds` (defined in the <sys/msg.h> header). Operations on the buffer are defined by *cmd*. Refer to the "Description" section below.

Return Values

0 Success.

-1 An error occurred, and `errno` is set to indicate the error condition.

Description

The `msgctl()` function provides message control operations on the message queue and data structure associated with *msqid*. Control operations are defined by the *cmd* parameter. Following are valid commands to be passed in *cmd*:

`IPC_RMID` Deallocate the message queue identifier specified by *msqid* and purge the message queue and data structure associated with it. The calling process must have either MPE/iX SM capability, or be the owner or creator of *msqid* (have an effective user ID equal to the value of either the `msg_perm.uid` (owner) or `msg_perm.cuid` (creator) fields in the data structure associated with *msqid*).

`IPC_SET` Copy data from the following fields of the `msqid_ds` structure pointed to by *buffer* to the corresponding fields in the data structure associated with *msqid*:

- `msg_perm.uid` (owner user ID)
- `msg_perm.gid` (owner group ID)
- Low order 9 bits of `msg_perm.mode`
- `msg_qbytes`

The calling process must have either MPE/iX SM capability or an effective user ID equal to the value of either the `msg_perm.uid` or `msg_perm.cuid` fields in the data structure associated with *msqid*. The caller must have MPE/iX SM capability to increase the value of the `msg_qbytes` field of the data structure associated with *msqid*.

`IPC_STAT` Copy all data from the structure associated with *msqid* to the data structure pointed to by *buffer*.

Implementation Considerations

None.

Errors

If an error occurs, `errno` is set to one of the following values.

`EACCES`	CAUSE	*cmd* specifies `IPC_STAT` and the calling process does not have read permission.
	ACTION	Ensure that the calling process has read permission for the *msqid*.
`EFAULT`	CAUSE	The system detected a NULL or bad address in attempting to use the *buffer* argument.
	ACTION	Check to see if the pointer is correctly initialized.
`EINVAL`	CAUSE	*msqid* is not a valid message queue identifier, or *cmd* is not a valid command.
	ACTION	Check that the *msqid* parameter is valid and the associated message queue has not been removed, or that *cmd* is valid.
`EPERM`	CAUSE	One of the following: ■ *cmd* specifies `IPC_RMID` or `IPC_SET` and the calling process does not have either MPE/iX SM capability or an effective user ID equal to the value of either the **`msg_perm.uid`** or **`msg_perm.cuid`** fields in the data structure associated with *msqid*. ■ *cmd* specifies `IPC_SET` and the calling process tried to increase the value of **`msg_qbytes`** without having MPE/iX SM capability.
	ACTION	Ensure that the calling process has the appropriate effective user ID or capability required to perform the requested command.
`ESYSERR`	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`msgget()`, `msgrcv()`, `msgsnd()`, SVID2 (Section 12)

msgget

Returns a message queue identifier.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key_t key, int msgflg);
```

Parameters

key — Either a user-defined key value to rendezvous with the message queue, or `IPC_PRIVATE`. If `IPC_PRIVATE` is specified, a new message queue is created, but other processes cannot rendezvous by *key*. Refer to the description of `ftok()` for details about obtaining user-defined key values

msgflg — Valid flags for this function are:

- `IPC_CREAT` — If a message queue is not already associated with *key*, a new message queue identifier is allocated and a message queue and data structure are associated with it. If a message queue identifier is already associated with *key*, and `IPC_EXCL` is not specified, `msgget()` returns the message queue identifier associated with *key*.
- `IPC_EXCL` — If specified with `IPC_CREAT`, `msgget()` returns an error if a message queue identifier is already associated with *key*.
- `MODE` — The lower nine bits of *msgflg* contain access permission bits (similar to the nine-bit mask found in file entries). They define access permissions for the owner, the group, and other users on the system.

Return Values

>0	Success. A message queue identifier is returned.
-1	An error occurred, and `errno` is set to indicate the error condition.

Description

The `msgget()` function returns a message queue identifier associated with the value passed in *key*. A new message queue identifier is allocated and a message queue and data structure are associated with it if:

- The value passed in *key* is equal to `IPC_PRIVATE`.
- The value passed in *key* does not already have a message queue identifier associated with it, and *msgflg* specifies `IPC_CREAT`.

The data structure associated with the new message queue identifier is initialized to the following values:

`msg_perm.cuid`	Effective user ID of the calling process (creator user ID)
`msg_perm.uid`	Effective user ID of the calling process (owner user ID)
`msg_perm.cgid`	Effective group ID of the calling process (creator group ID)
`msg_perm.gid`	Effective group ID of the calling process (owner group ID)
`msg_perm.mode`	Low-order 9 bits are set equal to the low-order 9 bits of *msgflg*
`msg_qnum`	Zero
`msg_lspid`	Zero
`msg_lrpid`	Zero
`msg_stime`	Zero
`msg_rtime`	Zero
`msg_ctime`	Current time
`msg_qbytes`	System limit

Implementation Considerations

The maximum size of a message is 65536 bytes.

An MPE/iX system manager can use the MPE/iX `SVIPC` utility to interactively configure:

- The maximum message size

- The maximum number of bytes on a message queue
- The total number of message queues allowed system wide

Refer to the section "Managing SVID IPC Services" for more information.

Errors

If an error occurs, `errno` is set to one of the following values.

`EACCES`	CAUSE	A message queue identifier exists for *key* and the calling process does not have permission (specified by the low-order 9 bits of *msgflg*).
	ACTION	Ensure that the calling process has appropriate permissions to access the existing message queue associated with *key*, or specify a unique *key* value to create a new message queue.
`EEXIST`	CAUSE	A message queue identifier exists for *key* and *msgflg* specifies both `IPC_CREATE` and `IPC_EXCL`.
	ACTION	To access the existing message queue associated with *key*, remove the `IPC_EXCL` option. Otherwise, a unique *key* value must be specified.
`ENOENT`	CAUSE	A message queue identifier does not exist for *key* and *msgflg* does not specify `IPC_CREATE`.
	ACTION	Specify `IPC_CREATE` to indicate a message queue should be created if one does not already exist for the specified *key* value.
`ENOSPC`	CAUSE	The number of message queue identifiers would exceed the system-defined limit.
	ACTION	A new message queue cannot be created unless a previously allocated message queue is removed.
`ESYSERR`	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`msgctl()`, `msgrcv()`, `msgsnd()`, SVID2 (Section 12)

msgrcv

Reads a message from a message queue.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv (int msqid, void *msgp, int msgsz,
            long msgtyp, int msgflg);
```

Parameters

msqid Passes message queue identifier returned by a call to `msgget()`.

msgp Passes a pointer to a buffer whose structure is similar to the *msgbuf* example template located in `<msg.h>`, for example:

```
struct example_msgbuf {

    long   mtype;                    /* message type          */
    char   mtext[your_buffer_size]; /* message text          */

};
```

where *your_buffer_size* is an integer specifying the size of the buffer. The `mtype` field stores the received message's type as specified by the process that sent the message. The `mtext` field stores the text of the message. The size of `mtext` is specified by the *msgsz* argument.

msgsz Passes the size, in bytes, of `mtext`. Valid values are from 0 to a system-defined limit.

msgtyp Passes a value specifying the type of message requested. Following are valid values and their meanings:

0 Read the first message on the queue.

>0	Read the first message on the queue whose type equals *msgtyp*.
<0	Read a message from the queue whose type is the lowest type of all messages in that queue that is less than or equal to the absolute value of *msgtyp*.

msgflg Passes a value defining what action to take if either a message specified by *msgtyp* is not found on the message queue or the message is too large to fit in the buffer. Flags are:

`IPC_NOWAIT`	The calling process is not suspended. Control returns immediately with `errno` set to `ENOMSG`.
`MSG_NOERROR`	If the message to receive is larger than *msgsz*, the message is truncated to *msgsz* bytes. No error indication is given.

If *msgflg* does not specify `IPC_NOWAIT`, the calling process suspends execution until a message satisfying the *msgtyp* specifications is placed on the queue. When this occurs, control returns to `msgrcv()`. If `MSG_NOERROR` is not set, and the selected message is larger than the buffer pointed to by *msgp*, `msgrcv()` returns an error and sets `errno` to `E2BIG`.

Return Values

>=0	Success. The number of bytes actually placed into the `mtext` field of the data structure pointed to by *msgp* is returned.
-1	An error occurred, and `errno` is set to indicate the error condition.

Description

The `msgrcv()` function reads a message from the message queue specified by *msqid* and places it in the buffer pointed to by *msgp*.

If the `MSG_NOERROR` option is set in *msgflg*, the received message is truncated to *msgsz* bytes if it is larger than *msgsz*. The truncated part of the message is lost and no indication of the truncation is given.

If the calling process is suspended waiting for a message, the following conditions will cause `msgrcv()` to return an error and set `errno` to indicate the error condition.

- The message queue specified by *msqid* is removed from the system
- The calling process receives a signal that is to be caught.

If `msgrcv()` is successful, the following fields of the data structure associated with *msqid* are updated to the indicated values:

`msg_qnum`	Decremented by 1
`msg_lrpid`	PID of the calling process
`msg_rtime`	Current time

Implementation Considerations

If a process suspended during execution of `msgrcv()` receives a signal, control returns to the user with `errno` set to `EINTR`. Disabled signals are ignored.

Errors

If an error occurs, `errno` is set to one of the following values.

`E2BIG`	CAUSE	*msgsz* is less than the size of the message and *msgflg* does not specify `MSG_NOERROR`.
	ACTION	Increase the *msgsz* parameter and associated buffer space, or specify the `MSG_NOERROR` option to allow truncation of the received message.
`EACCES`	CAUSE	The calling process does not have permission.
	ACTION	Ensure that the calling process has read access for the message queue.
`EFAULT`	CAUSE	The system detected a NULL or bad address in attempting to use the *msgp* argument.
	ACTION	Check to see if the pointer is correctly initialized.
`EIDRM`	CAUSE	The message queue specified by *msqid* was removed while the process was suspended in `msgrcv()`.
	ACTION	None.
`EINTR`	CAUSE	A process waited in `msgrcv()` was interrupted by a signal.
	ACTION	Application dependent.
`EINVAL`	CAUSE	*msqid* is not a valid message queue identifier, or *msgsz* is less than 0 or greater than the system-defined limit.
	ACTION	Check the *msqid* to make sure it is valid and the message queue has not been removed from the system. Verify that a positive *msgsz* was specified that does not exceed the currently configured limit.
`ENOMSG`	CAUSE	The specified message queue does not contain a message of the type specified in `mtype` and *msgflg* specifies `IPC_NOWAIT`.
	ACTION	None. Application dependent. The receive operation can be retried.
`ESYSERR`	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`msgctl()`, `msgget()`, `msgsnd()`, SVID2 (Section 12)

msgsnd

Sends a message to a message queue.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd (int msqid, void *msgp, int msgsz,
            int msgflg);
```

Parameters

msqid Passes a message queue identifier returned by a call to `msgget()`.

msgp Passes a pointer to a buffer whose structure is similar to the *msgbuf* example template located in `<msg.h>`, for example:

```
struct example_msgbuf {

    long   mtype;                   /* message type          */
    char   mtext[your_buffer_size]; /* message text          */

    };
```

where *your_buffer_size* is an integer specifying the size of the buffer. The `mtype` field stores a positive integer value that can be used by a receiving process for message selection (refer to `msgrcv()`). The `mtext` field stores the text of the message. The size of `mtext` is specified by the *msgsz* argument.

msgsz The size, in bytes, of `mtext`. Valid values are from 0 to a system-defined limit.

msgflg Passes a value defining what action to take if the number of bytes passed would cause the size of the specified queue to exceed the system-defined limit (`msq_bytes`), the total size of message data on this queue would exceed the system-defined limit, or the system-wide message buffer pool is temporarily depleted due to the amount of data queued to all message queues.

Flags are:

`IPC_NOWAIT`	The calling process is not suspended. Control returns immediately with `errno` set to `EAGAIN`.

If *msgflg* does not specify `IPC_NOWAIT` and one of the previous conditions would occur, the calling process suspends execution. When the condition that caused the suspension no longer exists, `msgsnd()` continues execution.

Return Values

0	Success.
-1	An error occurred, and `errno` is set to indicate the error condition.

Description

The `msgsnd()` function sends a message stored in the buffer pointed to by *msgp* to the message queue associated with *msqid*.

If the message queue associated with *msqid* is removed from the system while the calling process is suspended waiting to send a message, `msgsnd()` returns an error and sets `errno to EIDRM` .

If `msgsnd()` is successful, the following fields of the data structure associated with *msqid* are updated to the indicated values:

`msg_qnum`	Incremented by 1
`msg_lrpid`	PID of the calling process
`msg_rtime`	Current time

Implementation Considerations

If a process suspended during execution of `msgsnd()` receives a signal, control returns to the user with `errno` set to `EINTR`. Disabled signals are ignored.

Errors

If an error occurs, `errno` is set to one of the following values.

EACCES	CAUSE	The calling process does not have permission.
	ACTION	Ensure that the calling process has write permission for the message queue.
EAGAIN	CAUSE	*msgflg* specifies `IPC_NOWAIT` and either the number of bytes passed would cause the size of the specified queue to exceed the system-defined limit, the total size of message data on this queue would exceed the system-defined limit, or the system-wide message buffer pool is temporarily depleted due to the amount of data queued to all message queues.
	ACTION	None. Application dependent. The send operation can be retried later.
EFAULT	CAUSE	The system detected a NULL or bad address in attempting to use the *msgp* argument.
	ACTION	Check to see if the pointer is correctly initialized.
EIDRM	CAUSE	The message queue specified by *msqid* was removed while `msgsnd()` was waiting on a message.
	ACTION	None.
EINTR	CAUSE	`msgsnd()` was interrupted by a signal.
	ACTION	None. Application dependent.

`EINVAL`	CAUSE	*msqid* is not a valid message queue identifier, or *msgsz* is less than 0 or greater than the system-defined limit, or **`mtype`** is less than 0.
	ACTION	Check the parameters to make sure the *msqid* is valid and has not been removed from the system, and that the *msgsz* and **`mtype`** are within valid ranges.
`ESYSERR`	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`msgctl()`, `msgget()`, `msgrcv()`, SVID2 (Section 12)

semctl

Provides semaphore control operations.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (int semid, int semnum, int cmd,
            semun semarg);
```

Parameters

semid Passes a semaphore identifier returned by a call to `semget()`.

semnum Passes a value indicating a particular semaphore in the semaphore set certain commands specified in *cmd* will apply to, if applicable.

cmd Passes a command defining the control operation to perform. Valid commands are defined in the "Description" section below.

semarg Passes an argument containing information about the semaphore set. Operations using *semarg* are defined by *cmd*. Refer to the "Description" section below. The argument passed must be of type `union semun`, having the following structure:

```
union semun {
               int val;
               struct semid_ds *buf;
               ushort *array;
             }semarg;
```

Return Values

>=0	Success. The value returned depends on the command passed in *cmd*. Refer to the list below of possible return values and their meanings.
-1	An error occurred, and `errno` is set to indicate the error condition.

Upon successful completion, the `semctl()` function returns one of the following values depending on the command passed in *cmd*:

Command	**Return Value**
`GETVAL`	The semaphore value of the semaphore specified by *semid* and *semnum*.
`GETNCNT`	The number of processes waiting for the semaphore value of the semaphore specified by *semid* and *semnum* to become greater than 0.
`GETZCNT`	The number of processes waiting for the semaphore value of the semaphore specified by *semid* and *semnum* to become 0.
`GETPID`	The PID of the process that last modified the semaphore specified by *semid* and *semnum*.
All others	0

Description

The `semctl()` function provides semaphore control operations on the semaphore set and data structure associated with the semaphore identifier passed in *semid*. Control operations are defined by *cmd*. Following are valid commands to be passed in *cmd* and the resulting operations:

Command	**Operation**
`IPC_RMID`	Deallocate the semaphore identifier specified by *semid* and purge the semaphore set and data structure associated with it. The calling process must have either MPE/iX SM capability or an effective user ID equal to the value of either the `sem_perm.uid` or `sem_perm.cuid` fields in the data structure associated with *semid*.

`IPC_SET` Copy data from the following fields of the `semid_ds` structure (defined in the <sys/sem.h> header) pointed to by *semarg.buf* to the corresponding fields in the data structure associated with *semid*:

- `sem_perm.uid` (owner user ID)
- `sem_perm.gid` (owner group ID)
- Low order 9 bits of `sem_perm.mode`

The calling process must have either MPE/iX SM capability or an effective user ID equal to the value of either the `sem_perm.uid` or `sem_perm.cuid` fields in the data structure associated with *semid*.

`IPC_STAT` Copy all data from the data structure associated with *semid* to the data structure pointed to by *semarg.buf*. The structure `semid_ds` is defined in the <sys/sem.h> header. The calling process must have read permission.

`GETVAL` Return the semaphore value of the semaphore specified by *semid* and *semnum*. The calling process must have read permission.

`SETVAL` Set the semaphore value of the semaphore specified by *semid* and *semnum* to *semarg.val* (must be >=0). This command clears in all processes the semaphore adjust value corresponding to the specified semaphore. The calling process must have write permission.

`GETPID` Return the PID of the process that last modified the semaphore specified by *semid* and *semnum*. The calling process must have read permission.

`GETNCNT` Return the number of processes waiting for the semaphore value of the semaphore specified by *semid* and *semnum* to become greater than zero. The calling process must have read permission.

`GETZCNT` Return the number of processes waiting for the semaphore value of the semaphore specified by *semid* and *semnum* to become 0. The calling process must have read permission.

`GETALL` — Copy the semaphore values of all semaphores associated with *semid* to the array pointed to by *semarg.array*. The calling process must have read permission.

`SETALL` — Set the semaphore values of all semaphores to the values specified in the array pointed to by *arg.array* (must be >=0). This command clears in all processes the semaphore adjust value corresponding to the specified semaphore. The calling process must have write permission.

Implementation Considerations

None.

Errors

If an error occurs, `errno` is set to one of the following values.

EACCES	CAUSE	The calling process does not have permission.
	ACTION	Ensure that the process has the required permissions to perform the specified *cmd*.
EFAULT	CAUSE	The system detected a NULL or bad address in attempting to use either the *semarg.buf* or *semarg.array* arguments.
	ACTION	Check the *semarg* parameter and make sure it is properly defined.
EINVAL	CAUSE	*semid* is not a valid semaphore identifier, or *cmd* is not a valid command, or *semnum* is less than zero or greater than or equal to the value stored in the **sem_nsems** field in the data structure associated with *semid*, or the values of **SETVAL** or **SETALL** are out of range.
	ACTION	Check the parameters to make sure a valid *semid* was specified and the semaphore set was not removed from the system, a valid *cmd* was specified, *semnum* references a semaphore that exists in this semaphore set, or **SETVAL** and **SETALL** values are in range.
EPERM	CAUSE	*cmd* specifies **IPC_RMID** or **IPC_SET** and the calling process does not have either MPE/iX SM capability or an effective user ID equal to the value of either the **sem_perm.uid** or **sem_perm.cuid** fields in the data structure associated with *semid*.
	ACTION	Ensure that the calling process has the appropriate effective user ID or the appropriate capabilities to perform the specified *cmd*.
ERANGE	CAUSE	*cmd* specifies either **SETVAL** or **SETALL** and the resulting semaphore value would be greater than the system-defined limit.
	ACTION	Ensure that the semaphore value(s) specified are within the system-defined range.
ESYSERR	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`semget()`, `semop()`, SVID2 (Section 12)

semget

Returns a semaphore identifier.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget (key_t key, int nsems, int semflg);
```

Parameters

key — Either a user-defined key value to rendezvous with the semaphore set, or `IPC_PRIVATE`. If `IPC_PRIVATE` is specified, a new semaphore set is created, but other processes cannot rendezvous by *key*. Refer to the description of `ftok()` for details about obtaining user-defined key values

nsems — The number of semaphores in the set. The maximum number of semaphores per set is 4096.

semflg — Valid flags for this function are:

- `IPC_CREAT` — If a semaphore set is not already associated with *key*, a new semaphore identifier is allocated and a semaphore set and data structure are associated with it. If a semaphore identifier is already associated with *key*, and `IPC_EXCL` is not specified, `semget()` returns the semaphore identifier currently associated with *key*.
- `IPC_EXCL` — If specified with `IPC_CREAT`, `semget()` returns an error if a semaphore identifier is currently associated with *key*.
- `MODE` — The lower nine bits of *semflg* contain the access permission bits (similar to the nine-bit mask found in file entries). They define the access rights for the owner, the group, and other users on the system.

Return Values

>=0	Success. A semaphore identifier is returned.
-1	An error occurred, and `errno` is set to indicate the error condition.

Description

The `semget()` function returns a semaphore identifier associated with the value passed in *key*. A semaphore identifier and the associated data structure and semaphore set containing *nsems* semaphores are created for *key* if one of the following conditions is true:

- The value passed in *key* is equal to `IPC_PRIVATE`.
- The value passed in *key* does not already have a semaphore identifier associated with it, and the *semflg* specifies `IPC_CREAT`.

The data structure associated with the new semaphore identifier is initialized to the following values:

`sem_perm.cuid`	Effective user ID of the calling process (creator user ID)
`sem_perm.uid`	Effective user ID of the calling process (owner user ID)
`sem_perm.cgid`	Effective group ID of the calling process (creator group ID)
`sem_perm.gid`	Effective group ID of the calling process (owner group ID)
`sem_perm.mode`	Low-order 9 bits are set equal to the low-order 9 bits of *semflg*
`sem_nsems`	The value of *nsems*
`sem_otime`	0
`sem_ctime`	Current time

Implementation Considerations

The maximum number of semaphores per set is 4096.

An MPE/iX system manager can use the MPE/iX `SVIPC` utility to interactively configure:

- The maximum number of semaphores in a semaphore set
- The maximum value for a semaphore
- The maximum semaphore operation value
- The maximum semaphore adjust value

- The maximum number of semaphore sets allowed system wide
- The maximum number of semaphore operations allowed per `semop()` call

Refer to the section “Managing SVID IPC Services” for more information.

Errors

If an error occurs, `errno` is set to one of the following values.

`EACCES`	CAUSE	A semaphore identifier exists for *key* and the calling process does not have permission (specified by the low-order 9 bits of *semflg*).
	ACTION	Ensure the calling process has access permissions required to access the existing semaphore identifier.
`EEXIST`	CAUSE	A semaphore identifier exists for *key* and *semflg* specifies both `IPC_CREAT` and `IPC_EXCL`.
	ACTION	To access the existing identifier for *key*, retry the operation without the `IPC_EXCL` option. To create a new semaphore set with `IPC_EXCL`, a unique key must be specified.
`EINVAL`	CAUSE	*nsems* is either less 1 or greater than the system-defined limit, or a semaphore identifier exists for *key* and the number of semaphores in the set is less than *nsems* and *nsems* is not equal to 0.
	ACTION	If accessing an existing semaphore set, make sure the *nsems* value does not exceed the number of semaphores in the existing set. If creating a new semaphore set, make sure *nsems* does not exceed the system-defined limit.
`ENOENT`	CAUSE	A semaphore identifier does not exist for *key* and *semflg* does not specify `IPC_CREAT`.
	ACTION	If attempting to access an existing semaphore set, make sure the right *key* value was specified. If a new semaphore set should be created for that *key* if none exists, then make sure `IPC_CREAT` is specified.
`ENOSPC`	CAUSE	The number of semaphore identifiers would exceed the system-defined limit.
	ACTION	None. The operation can be retried if another semaphore identifier is removed from the system.

ESYSERR	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

semctl(), semop(), SVID2 (Section 12)

semop

Performs operations on a set of semaphores.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop (int semid, struct sembuf *sops, int nsops);
```

Parameters

semid Passes a semaphore identifier returned by a `semget()` call.

sops Passes a pointer to an array of semaphore operation structures where each element is of type `struct sembuf` (defined in the `<sys/sem.h>` header). Semaphore operation structures define operations to perform on the semaphore set. For details on using semaphore operation structures, refer to the "Description" section below.

nsops Passes the number of valid semaphore operation structures in the array pointed to by *sops*.

Return Values

0 Success.

-1 An error occurred, and `errno` is set to indicate the error condition.

Description

The `semop()` function performs operations on the set of semaphores associated with the semaphore identifier specified by *semid*.

The *sops* argument points to an array where each of *nsops* elements contains a semaphore operation structure. Each semaphore operation specified by the `sem_op` field is performed on the semaphore specified by `sem_num`. The operation is further defined by the `sem_flg` field. No semaphore operations are performed until blocking conditions on all of the semaphores specified in the array are removed.

If the value of `sem_op` is less than 0 and the calling process has write permission, one of the following operations occurs depending upon the current semaphore value and the value of `sem_flag`:

Operations when `sem_op` <0

Semaphore Value	`sem_flag` Value	Operation
>= absolute value of `sem_op`	0 or `IPC_NOWAIT`	A new semaphore value is calculated as the result of subtracting the absolute value of `sem_op` from the current semaphore value. The call to `semop()` returns successfully to the calling process.
>= absolute value of `sem_op`	`SEM_UNDO`	A new semaphore value is calculated as the result of subtracting the absolute value of `sem_op` from the current semaphore value. The absolute value of `sem_op` is added to the calling process's semaphore adjust value of the specified semaphore. The call to `semop()` returns successfully to the calling process.
< absolute value of `sem_op`	`IPC_NOWAIT`	`semop()` returns -1, sets `errno` to `EAGAIN`, and returns control to the calling process.
< absolute value of `sem_op`	`IPC_NOWAIT` not specified	The `semncnt` field is incremented, indicating the number of processes waiting for the semaphore value of the specified semaphore to become greater than zero. Process execution is suspended until one of the following conditions occurs: ▪ The semaphore value becomes greater than or equal to the absolute value of `sem_op`. When this occurs, the `semncnt` field is decremented by 1 and execution continues as described above when semaphore value >= absolute value of `sem_op`. ▪ The semaphore identifier is removed from the system. `semop()` returns with a value of -1 and `errno` is set to `EIRDM`. ▪ A signal is caught by the suspended process. When this occurs, the `semncnt` field is decremented by 1 and the calling process resumes execution in the manner defined by the signal facility.

If the value of `sem_op` is equal to 0 and the calling process has read permission, one of the following operations occurs depending upon the current semaphore value and the value of `sem_flag`:

Operations when `sem_op`=0

Semaphore Value	`sem_flag` Value	Operation
0	Any value	`semop()` executes the next semaphore operation in the array pointed to by *semops*, or returns successfully to the calling process if there are no more valid semaphore operations.
<>0	`IPC_NOWAIT`	`semop()` returns -1, sets `errno` to `EAGAIN`, and returns control to the calling process.
<>0	0 or `SEM_UNDO`	The `semzcnt` field is incremented, indicating the number of processes waiting for the semaphore value of the specified semaphore to become zero. Process execution is suspended until one of the following conditions occurs: ■ The semaphore value becomes zero. When this occurs, the `semzcnt` field is decremented by 1 and execution continues as described above when semaphore value = 0. ■ The specified semaphore identifier is removed from the system. `semop()` returns with a value of -1 and `errno` is set to `EIRDM`. ■ A signal is caught by the suspended process. When this occurs, the `semncnt` field is decremented by 1 and the calling process resumes execution in the manner defined by the signal facility.

If the value of sem_op is greater than 0 and the calling process has write permission, one of the following operations occurs depending upon the current semaphore value and the value of sem_flag.

Operations when sem_op>0

Semaphore Value	sem_flag Value	Operation
Any value	SEM_UNDO not specified	A new semaphore value is calculated as the result of adding the value of sem_op to the current semaphore value of the specified semaphore. The call to semop() returns successfully to the calling process.
Any value	SEM_UNDO	A new semaphore value is calculated as the result of adding the value of sem_op to the current semaphore value of the specified semaphore. The value of sem_op is subtracted from the calling process's semaphore adjust value of the specified semaphore. The call to semop() returns successfully to the calling process.

If semop() is successful, the value of sempid for each semaphore specified in the array pointed to by *sops* is set equal to the PID of the calling process. The value of sem_otime in the data structure associated with the semaphore identifier is set to the current time.

Implementation Considerations

If a process suspended during execution of semop() receives a signal, control returns to the user with errno set to EINTR. Disabled signals are ignored.

The maximum number of semaphore UNDO entries per process is 64. Semaphore adjust values can be maintained for up to 64 distinct semaphores (elements of semaphore sets).

An MPE/iX system manager can use the MPE/iX `SVIPC` utility to interactively configure:

- the maximum semaphore value
- the maximum semaphore adjust value
- the maximum `nsops` value
- The maximum `sem_op` value

Refer to the section "Managing SVID IPC Services" for more information.

Errors

If an error occurs, `errno` is set to one of the following values.

`E2BIG`	CAUSE	*nsops* specifies a value greater than the system-defined limit.
	ACTION	Check the *nsops* value and make sure it is within the system-defined range.
`EACCES`	CAUSE	The calling process does not have permission.
	ACTION	Ensure the process has write permission (to modify a semaphore value) or read permission (to test a semaphore for 0).
`EAGAIN`	CAUSE	`sem_flg` specifies `IPC_NOWAIT` and the calling process would suspend on the specified operation.
	ACTION	None. Application dependent.
`EFAULT`	CAUSE	The system detected a NULL or bad address in attempting to use the *sops* argument.
	ACTION	Check to see that the *sops* argument has been properly defined.
`EFBIG`	CAUSE	`sem_num` is either less than zero or greater than or equal to the number of semaphores in the semaphore set associated with *semid*.
	ACTION	Check the `sem_num` value to make sure it specifies a valid semaphore in the semaphore set identified by *semid*.

`EIDRM`	CAUSE	The semaphore set specified by *semid* was removed while `semop()` was suspended on a semaphore operation.
	ACTION	None.
`EINTR`	CAUSE	`semget()` was interrupted by a signal.
	ACTION	None. Application dependent.
`EINVAL`	CAUSE	*semid* is not a valid semaphore identifier, or the calling process requested a `SEM_UNDO` for a number of semaphores that would exceed the system-defined limit.
	ACTION	Check that *semid* specifies a valid semaphore identifier and that it has not been removed from the system.
`ENOSPC`	CAUSE	The number of maximum undo entries (64) for this process would exceed the system-defined limit.
	ACTION	None. There were no undo table entries available to record the `SEM_UNDO` information. Examine the application to see if `SEM_UNDO` is required for that many semaphores.
`ERANGE`	CAUSE	The resulting semaphore value or semaphore adjust value would exceed the system-defined limit.
	ACTION	None. Application dependent.
`ESYSERR`	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`semctl()`, `semget()`, SVID2 (Section 12)

shmat

Attaches the calling process to a shared memory area.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat (int shmid, char *shmaddr, int shmflg);
```

Parameters

shmid Passes a shared memory identifier returned by a `shmget()` call.

shmaddr Passes either 0 or a valid memory address. Set *shmaddr* to zero to attach a shared memory area to the address space of the calling process. Otherwise, set *shmaddr* to the data start address of a shared memory area that is already attached to another process.

shmflg Passes a value specifying that the calling process has read/write access to the attached shared memory area. It is not possible to attach for write-only access or read-only access.

Return Values

addr Success. `shmat()` returns the data area start address of the attached shared memory area.

-1 An error occurred, and `errno` is set to indicate the error condition.

Description

The `shmat()` function attaches the shared memory area associated with the shared memory identifier specified by *shmid* to the data area of the calling process.

If the shared memory area is already attached to another process, a non-zero value of *shmaddr* is accepted, provided the specified address is identical to the current attach address of the area. The area is attached for both reading and writing.

Implementation Considerations

The MPE/iX implementation of SVID IPC shared memory emulates the equivalent functionality on a Series 800 HP9000 computer system. A process cannot attach to the same *shmid* multiple times. The address must be the same in all processes. Specifying a different address results in an error.

When attaching to a shared memory area for the first time, *shmaddr* must be set to zero.

The `SHM_RND` and `SHMLBA` flags are not supported. Specifying `SHM_RND` results in an error, and `errno` is set to `EACCES`.

The maximum number of shared memory areas a process can attach to is 256.

When `fork()` is called, the child process inherits all shared memory areas to which the parent process is attached. When `exec()` is called, the shared memory attached to the calling process is not attached to the new process.

Errors

If an error occurs, `errno` is set to one of the following values.

EACCES	CAUSE	The calling process does not have permission, or SHM_RND was specified.
	ACTION	Ensure that the calling process has permission to access the area as requested, or do not specify the SHM_RND and SHMLBA flags.
EINVAL	CAUSE	*shmid* is not a valid shared memory identifier, or *shmaddr* is not zero and not equal to the current attach location for the shared memory area, or the calling process is already attached to the shared memory area.
	ACTION	Check that *shmid* is valid and has not been removed from the system. If there is no current attach location for the shared memory area, make sure *shmaddr* is zero. A process cannot attach more than once (concurrently) to the same shared memory area.
EMFILE	CAUSE	The number of shared memory areas attached to the calling process would exceed the system-defined limit.
	ACTION	None. The operation can be retried if the process detaches from another shared memory area to which it is currently attached.
ENOMEM	CAUSE	The available data space is not large enough to accommodate the shared memory area.
	ACTION	None. The operation can be retried later.
ESYSERR	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`shmctl()`, `shmdt()`, `shmget()`, SVID2 (Section 12)

shmctl

Performs control operations on a shared memory area.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (int shmid, int cmd, struct shmid_ds *buffer);
```

Parameters

shmid Passes a shared memory identifier returned by a call to `shmget()`.

cmd Passes a command defining the control operation to perform. Valid control codes are defined in the "Description" section below.

buffer Passes a pointer to a buffer of type `struct shmid_ds` (defined in the <sys/shm.h> header). Operations on the buffer are defined by *cmd*. Refer to the "Description" section below.

Return Values

0 Success.

-1 An error occurred, and `errno` is set to indicate the error condition.

Description

The `shmctl()` function performs control operations on *shmid* and its associated shared memory area and data structure. Control operations are defined by the *cmd* parameter. Following are valid commands to be passed in *cmd*:

`IPC_RMID` Deallocate the shared memory identifier specified by *shmid* and purge the shared memory area and data structure associated with it. If the shared memory area is attached to one or more processes the shared memory area *key* is changed to `IPC_PRIVATE` and the area is marked removed. The area is purged only when the last attached process detaches from it.

The calling process must have either MPE/iX SM capability or be the owner or creator of the shared memory area (have an effective user ID equal to the value of either the `shm_perm.uid` (owner) or `shm_perm.cuid` (creator) fields in the data structure associated with *shmid*).

`IPC_SET` Copy data from the following fields of the `shmid_ds` structure pointed to by *buffer* to the corresponding fields in the data structure associated with *shmid*:

- `shm_perm.uid` (owner user ID)
- `shm_perm.gid` (owner group ID)
- Low order 9 bits of `shm_perm.mode`

The calling process must have either MPE/iX SM capability or an effective user ID equal to the value of either the `shm_perm.uid` or `shm_perm.cuid` fields in data structure associated with *shmid*.

`IPC_STAT` Copy all data from the structure associated with *shmid* to the data structure pointed to by *buffer*.

Implementation Considerations

The `SHM_LOCK` and `SHM_UNLOCK` options of *cmd* are not implemented. A call to `shmctl()` with *cmd* set to either `SHM_LOCK` or `SHM_UNLOCK` results in an error.

Errors

If an error occurs, `errno` is set to one of the following values.

EACCES	CAUSE	*cmd* is set to **IPC_STAT** and the calling process does not have read permission.
	ACTION	Ensure that the calling process has read permission to the shared memory area.
EFAULT	CAUSE	The system detected a NULL or bad address in attempting to use the *buffer* argument.
	ACTION	Check to see if the pointer is properly initialized.
EINVAL	CAUSE	*shmid* is not a valid shared memory identifier, or *cmd* is not a valid command, or *cmd* specifies **SHM_UNLOCK** or **SHM_LOCK**.
	ACTION	Check that *shmid* is valid and that the identifier has not been removed from the system, and check that *cmd* specifies a valid, supported command.
ENOMEM	CAUSE	*cmd* specifies **SHM_LOCK** or the available data space is not large enough to accommodate the shared memory area.
	ACTION	None.
EPERM	CAUSE	*cmd* specifies **IPC_RMID** or **IPC_SET** and the calling process does not have either MPE/iX SM capability or an effective user ID equal to the value of either the **shm_perm.uid** (owner) or **shm_perm.cuid** (creator) fields in the data structure associated with *shmid*.
	ACTION	Ensure that the calling process has the appropriate effective user ID or the appropriate capabilities to perform the specified *cmd*.
ESYSERR	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`shmat()`, `shmdt()`, `shmget()`, SVID2 (Section 12)

shmdt

Detaches a process from a shared memory area.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmdt (char *shmaddr);
```

Parameters

shmaddr	Passes the address of the shared memory area (returned from a call to `shmat()`).

Return Values

0	Success.
-1	An error occurred, and `errno` is set to indicate the error condition.

Description

The `shmdt()` function detaches a shared memory area from the calling process's data area. The address of the shared memory area is specified by *shmaddr* (returned from a call to `shmat()`).

Implementation Considerations

None.

Errors

If an error occurs, `errno` is set to one of the following values.

`EINVAL`	CAUSE	*shmaddr* is not the data area start address of a shared memory area.
	ACTION	Check to see that *shmaddr* is equal to the value returned by a previous **`shmat()`** call.
`ESYSERR`	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`shmat()`, `shmctl()`, `shmget()`, SVID2 (Section 12)

shmget

Returns a shared memory identifier.

Syntax

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget (key_t key, int size, int shmflg);
```

Parameters

key — Passes either a user-defined key value to rendezvous with the shared memory area, or `IPC_PRIVATE`. If `IPC_PRIVATE` is specified, a new shared memory area is created, but other processes cannot rendezvous by *key*. Refer to the description of `ftok()` for details about obtaining user-defined key values

size — Passes the size, in bytes, of the shared memory area. The maximum shared memory area size is 256 megabytes.

shmflg — Valid flags for this function are:

`IPC_CREAT` — If a shared memory area is not already associated with *key*, a new shared memory area identifier is allocated and a shared memory area and data structure are associated with it. If a shared memory area is already associated with *key*, `shmctl()` returns the shared memory identifier associated with *key*.

`IPC_EXCL` — If specified with `IPC_CREAT`,`shmget()` returns an error if a shared memory identifier is already associated with *key*.

`MODE` — The lower nine bits of *shmflg* contain the access permission bits (similar to the nine bit

	mask found in file entries). They define access permissions for the owner, the group, and other users on the system.
`SHM_NO_PID`	Allocate a shared memory area without PID protection. (Refer to "Implementation Considerations" for more information about using this flag.)
`SHM_PRIV_ACCESS`	Allocate a shared memory area accessible only to a calling process that has MPE/iX user privileged mode (PM). (Refer to "Implementation Considerations" for more information about using this flag.)

Return Values

>=0	Success. A shared memory area identifier is returned.
-1	An error occurred, and `errno` is set to indicate the error condition.

Description

The `shmget()` function returns a shared memory identifier associated with the value passed in *key*. A new shared memory identifier is allocated and an associated data structure and shared memory area of *size* bytes are associated with it if:

- The value passed in *key* is equal to `IPC_PRIVATE`.
- The value passed in *key* does not already have a shared memory identifier associated with it, and *shmflg* specifies `IPC_CREAT`.

The data structure associated with the new shared memory identifier is initialized to the following values:

`shm_perm.cuid`	Effective user ID of the calling process (creator user ID)
`shm_perm.uid`	Effective user ID of the calling process (owner user ID)
`shm_perm.cgid`	Effective group ID of the calling process (creator group ID)
`shm_perm.gid`	Effective group ID of the calling process (owner group ID)
`shm_perm.mode`	Low-order 9 bits are set equal to the low-order 9 bits of *shmflg*

`shm_segsz`	Value passed in *size*
`msg_qnum`	Zero
`shm_lpid`	Zero
`shm_nattch`	Zero
`shm_atime`	Zero
`shm_dtime`	Zero
`shm_ctime`	Current time

Implementation Considerations

The maximum shared memory area size is 256 megabytes.

On MPE/iX, two flags, `SHM_NO_PID` and `SHM_PRIV_ACCESS` are available to the `shmget()` function that are not defined by SVID. (Refer to the description of *shmflg* above.)

Note The `SHM_NO_PID` and `SHM_PRIV_ACCESS` flags are available only on 900 Series HP 3000 computer systems. These two flags are not recommended for portable applications. Specifying these flags on a different computer system may produce unpredictable results. In addition, setting `SHM_NO_PID` increases the risk of data corruption, since the shared memory area will not be protected by normal MPE/iX data memory protection traps.

An MPE/iX system manager can use the MPE/iX `SVIPC` utility to interactively configure:

- The minimum and maximum size of the shared memory area
- The total number of shared memory areas allowed system wide.

Refer to the section "Managing SVID IPC Services" for more information.

Errors

If an error occurs, `errno` is set to one of the following values:

`EACCES`	CAUSE	A shared memory identifier exists for *key* but the calling process does not have permission (as specified by the low-order 9 bits of *shmflg*).
	ACTION	Ensure that the calling process has appropriate permissions to obtain access to the existing shared memory identifier.
`EEXIST`	CAUSE	A shared memory identifier exists for *key* and *shmflg* specifies both `IPC_CREATE` and `IPC_EXCL`.
	ACTION	To access the existing shared memory identifier, remove the `IPC_EXCL` option. Otherwise, a unique *key* value must be specified for a new shared memory identifier to be created.
`EINVAL`	CAUSE	*size* is either less than the system-defined minimum or greater than the system-defined maximum, or a shared memory identifier exists for *key* and the size of the shared memory area associated with it is less than *size* and *size* is not equal to zero.
	ACTION	Check to see that *size* is within the system-defined valid range, and that if a shared memory identifier already exists for *key*, that *size* is within that shared memory area's valid range.
`ENOENT`	CAUSE	A shared memory identifier does not exist for *key* and *shmflg* does not specify `IPC_CREATE`.
	ACTION	To create a shared memory area for *key* when one does not already exist, make sure `IPC_CREAT` is specified.
`ENOMEM`	CAUSE	The available data space is not large enough to create a shared memory identifier and associated shared memory area.
	ACTION	None. The operation can be retried if another shared memory identifier is removed from the system.
`ENOSPC`	CAUSE	The number of shared memory identifiers would exceed the system-defined limit.
	ACTION	None. The operation can be retried if another shared memory identifier is removed from the system.
`ESYSERR`	CAUSE	An operating system error occurred that does not map directly to any of the above errors.
	ACTION	Examine the MPE/iX process error stack for the type of system error.

See Also

`shmat()`, `shmctl()`, `shmdt()`, SVID2 (Section 12)

SVID IPC Header Descriptions

Headers required by SVID IPC provide MACRO, type, and structure definitions, as well as function prototypes. SVID IPC headers are located under the *to be provided* directory. In addition, the `<sys/types.h>` header, described in the *MPE/iX Developer's Kit Reference Manual* (36430-90001), defines additional features required by SVID IPC.

The following headers are required by SVID IPC:

- `<sys/types.h>`
- `<sys/ipc.h>`
- `<sys/msg.h>`
- `<sys/shm.h>`
- `<sys/sem.h>`

Headers required by each SVID IPC function are specified in the "Syntax" section of each function description. You must specify the headers in the indicated order. The following sections provide detailed descriptions of the SVID IPC headers.

sys/ipc.h

Macros

`IPC_CREAT`	Create entry if key doesn't exist
`IPC_EXCL`	Fail if key exists
`IPC_NOWAIT`	Error if request must wait
`IPC_PRIVATE (key_t)0`	Private key value
`IPC_RMID`	Remove identifier
`IPC_SET`	Set options
`IPC_STAT`	Get options

Functions

If `__STDC__` is defined:

```
extern int svipc_info( int, void *, void * );
extern int svipc_control( int, void *, void * );
```

If `__STDC__` is not defined:

```
extern int svipc_info( );
extern int svipc_control( );
```

Types

```
typedef long  key_t;
```

Structures

Common IPC access structure:

```
struct ipc_perm {

    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        */
    long    mode;               /* access modes              */
    long    seq;                /* slot usage sequence number*/
    key_t   key;                /* key                       */

    };
```

sys/msg.h

Macros

MSG_NOERROR	No error if big message
MSG_WWAIT	A writer is waiting on the queue
MSG_RWAIT	A reader is waiting on the queue

Functions

If __STDC__ is defined:

```
extern int msgget( key_t, int );
extern int msgctl( int, int, struct msqid_ds * );
extern int msgrcv( int, void *, int, long, int );
extern int msgsnd( int, void *, int, int );
```

If __STDC__ is not defined:

```
extern int msgget( );
extern int msgctl( );
extern int msgrcv( );
extern int msgsnd( );
```

Structures

Message queue control structure:

```
struct msqid_ds {

   struct ipc_perm   msg_perm;    /* msg_perm defined in sys/ipc.h    */
   void              *msg_first;  /* not used on MPE/iX               */
   void              *msg_last;   /* not used on MPE/iX               */
   int               msg_cbytes;  /* current # bytes on queue         */
   int               msg_qnum;    /* # of messages on queue           */
   int               msg_qbytes;  /* max # of bytes on queue          */
   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                */

   };
```

Message buffer template structure:

```
struct msgbuf {                  /*This is a sample template only   */

   long   mtype;                 /* message type                    */
   char   mtext[1];              /* message text                    */

   };
```

sys/sem.h

Macros

SEM_UNDO	Set up adjust on exit entry
GETNCNT	Get semncnt
GETPID	Get sempid
GETVAL	Get semval
GETALL	Get all semvals
GETZCNT	Get semzcnt
SETVAL	Set semval
SETALL	Set all semvals

Functions

If `__STDC__` is defined:

```
extern int semctl( int, int, int, union semun );
extern int semget( key_t, int, int );
extern int semop ( int, struct sembuf *, unsigned int );
```

If `__STDC__` is not defined:

```
extern int semctl( );
extern int semget( );
extern int semop ( );
```

Structures

Semaphore set id data structure:

```
struct semid_ds {

   struct ipc_perm   sem_perm;       /* operation permission struct   */
   void              *sem_base;      /* not used on MPE/iX            */
   int               sem_nsems;      /* # of semaphores in set        */
   time_t            sem_otime;      /* last semop time               */
   time_t            sem_ctime;      /* last change time              */
```

```
};
```

Semaphore semop array element template structure:

```
struct sembuf {

   int        sem_num;               /* semaphore #                   */
   int        sem_op;                /* semaphore operation           */
   long       sem_flg;               /* operation flags               */

   };
```

sys/shm.h

Macros

SHM_NO_PID	No pid protection - MPE/iX only
SHM_PRIV_ACCESS	Privileged area - MPE/iX only
SHM_RDONLY	Read access only
SHM_RND	Round attach address (not implemented)
SHM_DEST	Delete when attach = 0 (not implemented)
SHM_CLEAR	Clear on first attach (not implemented)
SHM_LOCK	Not implemented
SHM_UNLOCK	Not implemented

Functions

If __STDC__ is defined:

```
extern char *shmat( int, char *m, int );
extern int  shmctl( int, int, struct shmid_ds * );
extern int  shmdt ( char * );
extern int  shmget( key_t, int, int );
```

If __STDC__ is not defined:

```
extern char *shmat( );
extern int  shmctl( );
extern int  shmdt ( );
extern int  shmget( );
```

Structures

Shared memory ID control structure:

```
struct shmid_ds {

    struct ipc_perm  shm_perm;       /* permission structure         */
    int              shm_segsz;      /* segment size                 */
    void             *shm_ptbl;      /* not used on MPE/iX           */
    pid_t            shm_lpid;       /* pid of last shmop call       */
```

```
pid_t           shm_cpid;       /* pid of last change           */
int             shm_nattch;     /* attached users               */
int             shm_cnattch;    /* in memory attached users ??  */
time_t          shm_atime;      /* last shmat time              */
time_t          shm_dtime;      /* last shmdt time              */
time_t          shm_ctime;      /* last change time             */

void            *shm_ptr;       /* pointer to the shm area.     */
};
```

3

TERMINFO Database

Introduction

The `TERMINFO` database describes terminal and printer capabilities. A wide range of capabilities can be defined that include, for example, the number of lines and columns for the device, whether or not the terminal wraps at the right margin, or what character sequence causes a carriage return. The database is used by screen-oriented programs such as VI or `CURSES` programs. By using `TERMINFO` to handle the capabilities of individual devices, a program can work with a variety of devices without any changes to the code.

The `TERMINFO` descriptions are located in the directory pointed to by the environment variable `TERMINFO`. The default directory is `/usr/lib/terminfo`.

Note There are several hundred terminal descriptions in the `TERMINFO` database. Hewlett-Packard only explicitly supports the following two terminal descriptions:

- `hp2392a`
- `ansi`

The majority of terminals used by Hewlett-Packard customers are compatible with one of these two descriptions. The other descriptions are available for you to use, but they are not supported.

TERMINFO Source File

One or more devices are described in a TERMINFO source file. This section describes the contents of the source file.

Syntax of Device Descriptions

Each device entry in the TERMINFO source file has the following format:

```
alias₁ | alias₂ | ... | aliasₙ | fullname,
     capability₁, capability₂,
     .
     .
     .
     capabilityₙ,
```

The first line in the device description is called the header; it must start in column one of the file. The header contains the commonly-used aliases for the device being described and the full name, which by convention appears last on the line. The environmental variable TERM can be set to any one of the terminal aliases. Each name is separated by a vertical bar, (|). The aliases must be unique in the database. They must follow normal MPE/iX HFS naming conventions (avoid a hyphen in the alias name as it is used to append a suffix as described below). All lines in the file must end with a comma (,).

A sample header of the model 33 teletype follows:

```
33|tty33|tty|model 33 teletype
```

A special convention exists for naming terminals that have special hardware modes or user preferences (for example, a VT-100 with 132 columns). Attach a suffix to the alias name with a hyphen, as shown in the following example.

```
vt100-w|vt100 132 column,
```

More than one suffix can be used by concatenating them together. When using multiple suffixes, repeat the alias with the suffixes in the opposite order so the user does not have to remember which is the correct order. The following example shows a terminal in wide mode with no automatic margins:

```
vt100-w-nam|vt100-nam-w|vt100 132 column,
```

The suffix conventions used are shown in Table 3-1.

Table 3-1. Suffixes for Mode and User Preferences

Suffix	Meaning
`-am`	Auto margins (usually the default)
`-na`	No arrow keys (leave in local mode)
`-nam`	No auto margins
`-w`	Wide mode (more than 80 columns)
`-rv`	Reverse video
-*n*	Number of lines on the screen
-*n*`p`	Number of pages of memory

After the header come the descriptions of the capabilities, separated by commas (white space after the comma is ignored). Each line after the header is indented one or more spaces or tabs. An example that shows the syntax of the capabilities segment of the file follows:

```
dumb|Dumb terminal,
am, xon,
cols#80, it#8,
bel=^G, cr=\r, cudl=\n, ind=\n$<15>,
```

There are three types of capabilities: Boolean, numeric, and string. The first line in the example shows Boolean capabilities; the second line shows numeric capabilities; and the third shows string capabilities.

The Boolean capabilities indicate the presence or absence of a capability. They take no arguments. The terminal in the example has automatic margins (`am`) and uses the `XON/XOFF` handshaking protocols (`xon`).

The numeric capabilities show size, spacing, or some other measurement. The capability is followed immediately by a pound sign (`#`) character and a positive integer. The terminal in the example has a screen with 80 columns (`cols#80`) and tab stops initially set to every 8 characters (`it#8`).

The string capabilities describe a terminal operation. The capability is followed immediately by an equals sign (`=`), and the string that performs the operation.

The terminal in the previous example beeps the terminal when sent a ^G sequence, performs a carriage return when sent a return character, moves the cursor down a line when sent a newline character, and scrolls forward from the bottom line of the screen when sent a newline character. (Control characters are entered in the device description as a caret (^), followed by a letter, as opposed to entered as the actual control character.)

Padding

Padding is used to delay further output to terminals that need extra time to process the current command. Some terminals use the XON/XOFF protocol instead of padding to tell the sending computer not to send the next command until the terminal is ready to receive it. Padding can still be used with the XON/OFF protocol so that programs can calculate the speed of functions. Padding can be specified for all string capabilities with the exception of input capabilities (names preceded with key_).

Padding is specified by a dollar sign ($) followed by a number enclosed within angle brackets (for example, $<15>). A forward slash (/) after the number specifies that padding is mandatory; that is, it should be applied regardless of the XON/XOFF setting. An asterisk (*) after the number specifies proportional padding; that is, it is applied to each line affected.

The syntax of padding specifications is summarized in Table 3-2.

Table 3-2. Syntax of Padding Specifications

Padding Syntax	Meaning
$<|n	Indicates a delay in *n* milliseconds
$<|n/	Indicates the delay is mandatory
$<|n*	Indicates the padding to be applied for each line affected
$<|n/*	Indicates the mandatory padding to be applied for each line affected

Note The tputs() routine performs the necessary padding for output.

Parameterized Strings

Strings that require run-time parameters are described using `printf`-like escapes (`%x`). Calculations are done on a stack using Reverse Polish Notation. Parameters are pushed onto the stack, manipulated in some way, and a result is output. The left-most operators are pushed first; for example, to subtract 10 from the first parameter, you would use:

```
%p1%{10}%-.
```

A complex example that describes cursor movement for the Wyse-50 follows:

```
cup=\E=%p1%'\s'%+%c%p2%'\s'%+%c,
```

This parameterized string is described in Table 3-3.

Table 3-3. Explanation of Parameter Description

Expression	Meaning
`\E=`	Send cursor addressing command ^[=
`%p1`	Push the first parameter onto stack
`%'\s'`	Push a space onto stack
`%+`	Pop the first two values on stack, add them, and push sum back on stack
`%c`	Pop the value on top of stack as an ASCII character; send to terminal
`%p2`	Push the second parameter onto stack
`%'\s'`	Push a space onto stack
`%+`	Pop the first two values on stack, add them, and push sum back on stack
`%c`	Pop the value on top of stack as ASCII character; send to terminal

Stack Operators

The stack operators are defined as follows:

`%%`	Outputs the % character.
`%char`	Pop and print character on top of stack.
`%[ [:]flags] [field_width[.precision] ] [doxXs]`	Pop the topmost value and output as specified by the `printf`-like format. Flags are [`- + #`] and space.

Note When using the `-` flag with `%[doxXs]`, a colon (`:`), must be put between the `%` and `-` to distinguish the flag from the binary `%-` operator, for example, `%:-16.16s`.

%pn	Push *n*th parameter onto stack where *n* is a number from 1 to 9.
%'c'	Push character constant c onto stack.
`%{nn}`	Push decimal constant nn onto the stack.
`%l`	Pop a pointer to a string and push the length of the string onto the stack.
`%+ %- %* %/ %m`	Pop the top two values; add (int2 + int1), subtract (int2 - int1), multiply (int2 * int1), divide (int2 / int1), or modulo (int2 mod int1), and push the result.
`%& %| %^`	Pop the top two values, perform bitwise AND (int2 & int1), bitwise OR (int2 | int1), or bitwise XOR (int2 ^ int1)
`%= %> < %A %O`	Pop the top two values, push `TRUE` or `FALSE` depending on whether operators are equal, second is greater than first, second is less than first, both are true, either are true, respectively.
`%!`	Pop the top value and push its logical NOT.
`%~`	Pop the top value and push its bitwise NOT.

`%i`	For `ANSI` terminals, increment first parameter by one, if one parameter present, or increment first two parameters by one, if more than one parameter present.
`%? expr %t thenpart %e elsepart %;`	Execute thenpart if expr is `TRUE`; otherwise execute elsepart (elsepart is optional). Else-if's are possible: `%? c1 %t  b1 %e c2  %t b2 %e c3 %t b3, %e c4 %t b4 %e b5 %;` where `ci` are conditions and `bi` are bodies.

The following example describes more complex cursor movements using some of the stack operators described previously. For example, the `hp2392a` terminal needs to be sent the column before the row for cursor addressing. See Table 3-4 following the example for a complete explanation.

```
cup=\E&a%p2%dc%p1%dY,
```

Table 3-4. hp2392a Terminal Cursor Movement

Expression	Meaning
`\E&a`	Send ^[&a
`%p2\`	Push the second parameter onto the stack
`%d`	Print top of stack as decimal number
`c`	Send character c
`%p1`	Push the first parameter onto the stack
`%d`	Print top of stack as decimal number
`Y`	Send character Y

The following is an example of a cursor movement string for the `ANSI` terminal. The explanation of each argument is shown in Table 3-5.

```
cup=\E[%i%p1%d;%p2%dH,
```

Table 3-5. ANSI Terminal Cursor Movement

Expression	Meaning
`\E[`	Send ^[[
`%i`	Increment the first and second parameter by one
`%p1`	Push the first parameter onto the stack
`%d`	Print top of stack as a decimal number
`;`	Send character ;
`%p2`	Push the second parameter onto the stack
`%d`	Print top of stack as decimal number
`H`	Send character H

Creating Device Descriptions

The easiest way to create a new entry for a device is to find one that is similar, modify it, and compile it using the `tic` utility. The `tic` utility installs the new definition in the `TERMINFO` directory hierarchy. You can change the location of the directory hierarchy by redefining the `TERMINFO` environment variable.

You can test your description in small segments using `VI`. Keep in mind that a unusual device may not be adequately described by `TERMINFO` or adequately tested by `VI`.

A terminal can be defined as having certain capabilties that are equivalent to those of another terminal. These capabilities are then read from one terminal description into the other. The `use` capability names the terminal from which to read the capabilities. Any capabilities listed before the `use` string override those put in place by `use`. A capability that should not be used in the terminal desciption can be cancelled by typing an at sign (`@`) after the name of the capability. The following example bases the definition of a VT-100 terminal without automatic margins on a standard VT-100 terminal definition.

```
vt100-nam|VT100 without automatic margins,
```

```
am@, use=vt100,
```

A crude test for getting the right amount of padding for insert-line (if not documented) is to comment out `xon`, edit a large file at 9600 baud with `VI`, delete 16 or so lines from the middle of the screen, and press the u key several times quickly. If the display becomes corrupted, insert-line requires more padding.

Special Characters

Table 3-6 summarizes all the special characters sequences discussed to this point.

Table 3-6. Characters with Special Values

Character	Meaning
,	Separates capabilities
#	Precedes integer value in numeric capabilities
=	Separates string capability name from string sequence
@	Cancels capability
#	At the beginning of line, comments out the line
.	When directly preceding a capability, period comments it out
^x	Control x sequence
\E	Escape character
\e	Escape character
\n	Newline character
\r	Return character
\t	Tab character
\v	Vertical tab character
\b	Backspace character
\f	Formfeed character
\s	Space character
\l	Linefeed character
\a	Alert character

Table 3-6. Characters with Special Values (continued)

Character	Meaning
\xxx	Octal characters (must be three characters)
\,	Escapes a comma
\\	Escapes a backslash
\^	Escapes a caret
\:	Escapes a colon
\0	Escapes a null character
%x	String using format operator
$<n>	Indicates a delay in *n* milliseconds
$<n/>	Indicates the delay is mandatory
$<n*>	Indicates the padding required per affected line
$<n/*>	Indicates mandatory padding required per affected line

Names of Capabilities

Capability names are normally kept to two to five characters and normally comply with ANSI X3.64-1979. Each capability has a corresponding variable name used in the program to access it; for example, the variable defined for `am` is `auto_left_margin`.

The Boolean, number, and string capabilities are listed in the tables on the following pages. Within these tables, each `TERMINFO` capability name is listed, along with the equivalent `termcap` name, the variable name, and a brief description. The $\#i$ symbol that sometimes appears in the description section of the table refers to the ith parameter.

Additional tables sorted by variable name may be found in Table 3-20, Table 3-21, and Table 3-22 as a reference for programmers. Table 3-20 lists the capabilities by the Boolean variable name; Table 3-21 lists the capabilities by the numeric variable name, and Table 3-22 lists the capabilities by the string variable name.

Boolean Capabilities

Table 3-7 lists the Boolean capabilities.

Table 3-7. Boolean Capabilities

TInfo	TCap	Variable	Description
am	am	auto_right_margin	Terminal has automatic margins
bw	bw	auto_left_margin	cub1 wraps from column 0 to last column
ccc	cc	can_change	Terminal can redefine existing color
chts	HC	hard_cursor	Cursor is hard to see
cpix	YF	cpi_changes_res	Changing character pitch changes resolution
crxm	YB	cr_cancels_micro_mode	Using cr turns off micro mode
da	da	memory_above	Display may be retained above the screen
daisy	YC	has_print_wheel	Printer needs operator to change character set
db	db	memory_below	Display may be retained below the screen
eo	eo	erase_overstrike	Terminal can erase overstrikes with a blank
eslok	es	status_line_esc_ok	Escape can be used on the status line
gn	gn	generic_type	Generic line type (e.g. dialup, switch)
hc	hc	hard_copy	Hardcopy terminal
hls	hl	hue_lightness_saturation	Terminal uses only HLS color notation (Tektronix)
hs	hs	has_status_line	Terminal has extra status line
hz	hz	tilde_glitch	Hazeltine: cannot print tilde (˜)
in	in	insert_null_glitch	Insert mode distinguishes nulls
km	km	has_meta_key	Terminal has meta key (shift, sets parity bit)

Table 3-7. Boolean Capabilities (continued)

TInfo	TCap	Variable	Description
lpix	YG	lpi_changes_res	Changing line pitch changes resolution
mc5i	5i	prtr_silent	Printer will not echo on screen
mir	mi	move_insert_mode	Safe to move in insert mode
msgr	ms	move_standout_mode	Safe to move in standout mode
npc	NP	no_pad_char	Pad character does not exist
nrrmc	NR	non_rev_rmcup	smcup does not reverse rmcup
nxon	nx	needs_xon_xoff	Padding will not work, XON/XOFF required
os	os	over_strike	Terminal overstrikes
sam	YE	semi_auto_right_margin	Printing in last column causes cr
ul	ul	transparent_underline	Underline character overstrikes
xenl	xn	eat_newline_glitch	Newline ignored after 80 columns (Concept)
xhp	xs	ceol_standout_glitch	Standout not erased by overwriting (hp)
xhpa	YA	col_addr_glitch	Only positive motion for hpa/mhpa capabilities
xon	xo	xon_xoff	Terminal uses XON/XOFF handshaking
xsb	xb	no_esc_ctlc	Beehive (f 1=escape, f2=ctrl C)
xt	xt	dest_tabs_magic_smso	Tabs destructive, magic smso character (Teleray 1061)
xvpa	YD	row_addr_glitch	Only positive motion for vpa/mvpa capabilities

Numeric Capabilities

Table 3-8 lists the numeric capabilities.

Table 3-8. Numeric Capabilities

TInfo	TCap	Variable	Description
bufsz	Ya	buffer_capacity	Number of bytes buffered before printing
colors	Co	max_colors	Maximum number of colors on the screen
cols	co	columns	Number of columns in a line
it	it	init_tabs	Number of spaces between initial tabs
lh	lh	label_height	Number of rows in each label
lines	li	lines	Number of lines on screen or page
lm	lm	lines_of_memory	Lines of memory if > 0; 0 means unfixed
lw	lw	label_width	Number of columns in each label
maddr	Yd	max_micro_address	Maximum value in micro_ ... _address
mcs	Yf	micro_col_size	Character step size when in micro mode
mjump	Ye	max_micro_jump	Maximum value in parm_ ... _micro
mls	Yg	micro_line_size	Line step size when in micro mode
ncv	NC	no_color_video	Video attributes that cannot be used with colors
nlab	Nl	num_labels	Number of labels on screen (start at 1)
npins	yH	number_of_pins	Number of pins in print-head
orc	Yi	output_res_char	Horz. res. in units per character
orhi	Yk	output_res_horz_inch	Horz. res. in units per inch
orl	Yj	output_res_line	Vert. res. in units per line
orvi	Yl	output_res_vert_inch	Vert. res. in units per inch
pairs	pa	max_pairs	Maximum number of color pairs on the screen

Table 3-8. Numeric Capabilities (continued)

TInfo	TCap	Variable	Description
pb	pb	padding_baud_rate	Lowest baud rate where cr/nl padding needed
spinh	Yc	dot_horz_spacing	Spacing of dots horizontally in dots per inch
spinv	Yb	dot_vert_spacing	Spacing of pins vertically in pins per inch
vt	vt	virtual_termina	Virtual terminal number (CB/UNIX)
widcs	Yn	wide_char_size	Character step size when in double wide mode
wsl	ws	width_status_line	Number of columns in status line
xmc	sg	magic_cookie_glitch	Number of blank chars left by smso or rmso

String Capabilities

Table 3-9 lists the string capabilities.

Table 3-9. String Capabilities

TInfo	TCap	Variable	Description
acsc	ac	acs_chars	Graphics character set pairs aAbBcC-defn=vt100
bel	bl	bell	Produce audible signal (bell or beep)
blink	mb	enter_blink_mode	Turn on blinking
bold	md	enter_bold_mode	Turn on bold (extra bright) mode
cbt	bt	back_tab	Back tab
chr	ZC	change_res_horz	Change horizontal resolution
civis	vi	cursor_invisible	Make cursor invisible
clear	cl	clear_screen	Clear screen
cmdch	CC	command_character	Terminal settable command character in prototype
cnorm	ve	cursor_normal	Make cursor appear normal (undo `cvvis/civis`)
cpi	ZA	change_char_pitch	Change number of characters per inch
cr	cr	carriage_return	Move cursor to left edge of screen
csnm	Zy	char_set_names	List of character set names
csr	cs	change_scroll_region	Change to lines #1 through #2 (vt100)
cub	LE	parm_left_cursor	Move cursor left #1 spaces
cub1	le	cursor_left	Move cursor left one space
cud	DO	parm_down_cursor	Move cursor down #1 lines
cud1	do	cursor_down	Move cursor down one line
cuf	RI	parm_right_cursor	Move cursor right #1 spaces
cuf1	nd	cursor_right	Non-destructive space (cursor right)

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
cup	cm	cursor_address	Cursor motion to row #1 col #2
cuu	UP	parm_up_cursor	Move cursor up #1 lines
cuu1	up	cursor_up	Move cursor up
cvr	ZD	change_res_vert	Change vertical resolution
cvvis	vs	cursor_visible	Make cursor very visible
dch	DC	parm_dch	Delete #1 characters
dch1	dc	delete_character	Delete character
defc	ZE	define_char	Define a character in a character set
dim	mh	enter_dim_mode	Turn on half-bright mode
dl	DL	parm_delete_line	Delete #1 lines
dl1	dl	delete_line	Delete line
docr	Zw	these_cause_cr	Printing any of these chars causes cr
dsl	ds	dis_status_line	Disable status line
ech	ec	erase_chars	Erase #1 characters
ed	cd	clr_eos	Clear to end of display
el	ce	clr_eol	Clear to end of line
el1	cb	clr_bol	Clear to beginning of line, inclusive
enacs	eA	ena_acs	Enable alternate character set
ff	ff	form_feed	Hardcopy terminal page eject
flash	vb	flash_screen	Visible bell (may not move cursor)
fsl	fs	from_status_line	Return from status line
hd	hd	down_half_line	Half-line down (forward 1/2 linefeed)
home	ho	cursor_home	Home cursor (if no cup)

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
hpa	ch	column_address	Set cursor column
ht	ta	tab	Tab to next 8 space hardware tab stop
hts	st	set_tab	Set a tab in all rows, current column
hu	hu	up_half_line	Half-line up (reverse 1/2 linefeed)
ich	IC	parm_ich	Insert #1 blank characters
ich1	ic	insert_character	Insert character
if	if	init_file	Name of file containing is
il	AL	parm_insert_line	Add #1 new blank lines
il1	al	insert_line	Add new blank line
ind	sf	scroll_forward	Scroll text up
indn	SF	parm_index	Scroll forward #1 lines
initc	Ic	initialize_color	Initialize the definition of color
initp	Ip	initialize_pair	Initialize color pair
invis	mk	enter_secure_mode	Turn on blank mode (characters invisible)
ip	ip	insert_padding	Insert pad after character inserted
iprog	iP	init_prog	Pathname of program for initialization
is1=	i1	init_1string	Terminal initialization string
is2=	is	init_2string	Terminal initialization string
is3=	i3	init_3string	Terminal initialization string
kBEG	&9	key_sbeg	Sent by shift-beginning key
kCAN	&0	key_scancel	Sent by shift-cancel key
kCMD	*1	key_scommand	Sent by shift-command key
kCPY	*2	key_scopy	Sent by shift-copy key

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
kCRT	*3	key_screate	Sent by shift-create key
kDC	*4	key_sdc	Sent by shift-delete-char key
kDL	*5	key_sdl	Sent by shift-delete-line key
kEND	*7	key_send	Sent by shift-end key
kEOL	*8	key_seol	Sent by shift-eol key
kEXT	*9	key_sexit	Sent by shift-exit key
kFND	*0	key_sfind	Sent by shift-find key
kHLP	#1	key_shelp	Sent by shift-help key
kHOM	#2	key_shome	Sent by shift-home key
kIC	#3	key_sic	Sent by shift-insert-char key
kLFT	#4	key_sleft	Sent by shift-left key
kMOV	%b	key_smove	Sent by shift-move key
kMSG	%a	key_smessage	Sent by shift-message key
kNXT	%c	key_snext	Sent by shift-next key
kOPT	%d	key_soptions	Sent by shift-options key
kPRT	%f	key_sprint	Sent by shift-print key
kPRV	%e	key_sprevious	Sent by shift-prev key
kRDO	g	key_sredo	Sent by shift-redo key
kRES	%j	key_srsume	Sent by shift-resume key
kRIT	%i	key_sright	Sent by shift-right key
kRPL	%h	key_sreplace	Sent by shift-replace key
kSAV	!1	key_ssave	Sent by shift-save key
kSPD	!2	key_ssuspend	Sent by shift-suspend key
kUND	!3	key_sundo	Sent by shift-undo key

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
ka1	K1	key_a1	Upper left of keypad
ka3	K3	key_a3	Upper right of keypad
kb2	K2	key_b2	Center of keypad
kbeg	@1	key_beg	Sent by beginning key
kbs	kb	key_backspace	Sent by backspace key
kc1	K4	key_c1	Lower left of keypad
kc3	K5	key_c3	Lower right of keypad
kcan	@2	key_cancel	Sent by cancel key
kcbt	kB	key_btab	Sent by BackTab key
kclo	@3	key_close	Sent by close key
kclr	kC	key_clear	Sent by clear screen or erase key
kcmd	@4	key_command	Sent by command key
kcpy	@5	key_copy	Sent by copy key
kcreate	@6	key_create	Sent by create key
kctab	kt	key_ctab	Sent by clear-tab key
kcub1	kl	key_left	Sent by terminal left arrow key
kcud1	kd	key_down	Sent by terminal down arrow key
kcuf1	kr	key_right	Sent by terminal right arrow key
kcuu1	ku	key_up	Sent by terminal up arrow key
kdch1	kD	key_dc	Sent by delete character key
kdl1	kL	key_dl	Sent by delete line key
ked	kS	key_eos	Sent by clear-to-end-of-screen key
kel	kE	key_eol	Sent by clear-to-end-of-line key

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
kend	@7	key_end	Sent by end key
kent	@8	key_enter	Sent by enter/send key
kext	@9	key_exit	Sent by exit key
kf0	k0	key_f0	Sent by function key f0
kf1	k1	key_f1	Sent by function key f1
kf2	k2	key_f2	Sent by function key f2
kf3	k3	key_f3	Sent by function key f3
kf4	k4	key_f4	Sent by function key f4
kf5	k5	key_f5	Sent by function key f5
kf6	k6	key_f6	Sent by function key f6
kf7	k7	key_f7	Sent by function key f7
kf8	k8	key_f8	Sent by function key f8
kf9	k9	key_f9	Sent by function key f9
kf10	k;	key_f10	Sent by function key f10
kf11	F1	key_f11	Sent by function key 11
kf12	F2	key_f12	Sent by function key 12
kf13	F3	key_f13	Sent by function key 13
kf14	F4	key_f14	Sent by function key 14
kf15	F5	key_f15	Sent by function key 15
kf16	F6	key_f16	Sent by function key 16
kf17	F7	key_f17	Sent by function key 17
kf18	F8	key_f18	Sent by function key 18
kf19	F9	key_f19	Sent by function key 19

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
kf20	FA	key_f20	Sent by function key 20
kf21	FB	key_f21	Sent by function key 21
kf22	FC	key_f22	Sent by function key 22
kf23	FD	key_f23	Sent by function key 23
kf24	FE	key_f24	Sent by function key 24
kf25	FF	key_f25	Sent by function key 25
kf26	FG	key_f26	Sent by function key 26
kf27	FH	key_f27	Sent by function key 27
kf28	FI	key_f28	Sent by function key 28
kf29	FJ	key_f29	Sent by function key 29
kf30	FK	key_f30	Sent by function key 30
kf31	FL	key_f31	Sent by function key 31
kf32	FM	key_f32	Sent by function key 32
kf33	FN	key_f33	Sent by function key 33
kf34	FO	key_f34	Sent by function key 34
kf35	FP	key_f35	Sent by function key 35
kf36	FQ	key_f36	Sent by function key 36
kf37	FR	key_f37	Sent by function key 37
kf38	FS	key_f38	Sent by function key 38
kf39	FT	key_f39	Sent by function key 39
kf40	FU	key_f40	Sent by function key 40
kf41	FV	key_f41	Sent by function key 41
kf42	FW	key_f42	Sent by function key 42

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
kf43	FX	key_f43	Sent by function key 43
kf44	FY	key_f44	Sent by function key 44
kf45	FZ	key_f45	Sent by function key 45
kf46	Fa	key_f46	Sent by function key 46
kf47	Fb	key_f47	Sent by function key 47
kf48	Fc	key_f48	Sent by function key 48
kf49	Fd	key_f49	Sent by function key 49
kf50	Fe	key_f50	Sent by function key 50
kf51	Ff	key_f51	Sent by function key 51
kf52	Fg	key_f52	Sent by function key 52
kf53	Fh	key_f53	Sent by function key 53
kf54	Fi	key_f54	Sent by function key 54
kf55	Fj	key_f55	Sent by function key 55
kf56	Fk	key_f56	Sent by function key 56
kf57	Fl	key_f57	Sent by function key 57
kf58	Fm	key_f58	Sent by function key 58
kf59	Fn	key_f59	Sent by function key 59
kf60	Fo	key_f60	Sent by function key 60
kf61	Fp	key_f61	Sent by function key 61
kf62	Fq	key_f62	Sent by function key 62
kf63	Fr	key_f63	Sent by function key 63
kfnd	@0	key_find	Sent by find key
khlp	%1	key_help	Sent by help key

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
khome	kh	key_home	Sent by home key
khts	kT	key_stab	Sent by set-tab key
kich1	kI	key_ic	Sent by ins char/enter ins mode key
kil1	kA	key_il	Sent by insert line
kind	kF	key_sf	Sent by scroll-forward/down key
kll	kH	key_ll	Sent by home down key (lower left)
kmov	%4	key_move	Sent by move key
kmrk	%2	key_mark	Sent by mark key
kmsg	%3	key_message	Sent by message key
knp	kN	key_npage	Sent by next-page key
knxt	%5	key_next	Sent by next-object key
kopn	%6	key_open	Sent by open key
kopt	%7	key_options	Sent by options key
kpp	kP	key_ppage	Sent by previous-page key
kprt	%9	key_print	Sent by print or copy key
kprv	%8	key_previous	Sent by previous-object key
krdo	%0	key_redo	Sent by redo key
kref	&1	key_reference	Sent by reference key
kres	&5	key_resume	Sent by resume key
krfr	&2	key_refresh	Sent by refresh key
kri	kR	key_sr	Sent by scroll-backward/up key
krmir	kM	key_eic	Sent by rmir or smir in insert mode
krpl	&3	key_replace	Sent by replace key

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
krst	&4	key_restart	Sent by restart key
ksav	&6	key_save	Sent by save key
kslt	*6	key_select	Sent by select key
kspd	&7	key_suspend	Sent by suspend key
ktbc	ka	key_catab	Sent by clear-all-tabs key
kund	&8	key_undo	Sent by undo key
lf0	l0	lab_f0	Labels on function key f0 if not f0
lf1	l1	lab_f1	Labels on function key f1 if not f1
lf2	l2	lab_f2	Labels on function key f2 if not f2
lf3	l3	lab_f3	Labels on function key f3 if not f3
lf4	l4	lab_f4	Labels on function key f4 if not f4
lf5	l5	lab_f5	Labels on function key f5 if not f5
lf6	l6	lab_f6	Labels on function key f6 if not f6
lf7	l7	lab_f7	Labels on function key f7 if not f7
lf8	l8	lab_f8	Labels on function key f8 if not f8
lf9	l9	lab_f9	Labels on function key f9 if not f9
lf10	la	lab_f10	Labels on function key f10 if not f10
ll	ll	cursor_to_ll	Last line, first column (if no cup)
lpi	ZB	change_line_pitch	Change number of lines per inch
mc0	ps	print_screen	Print contents of the screen
mc4	pf	prtr_off	Turn off the printer
mc5	po	prtr_on	Turn on the printer
mc5p	p0	prtr_non	Turn on the printer for #1 bytes

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
mcub	Zg	parm_left_micro	Like parm_left_cursor
mcub1	Za	micro_left	Like cursor_left for micro adjustment
mcud	Zf	parm_down_micro	Like parm_down_cursor
mcud1	ZZ	micro_down	Like cursor_down for micro adjustment
mcuf	Zh	parm_right_micro	Like parm_right_cursor
mcuf1	Zb	micro_right	Like cursor_right for micro adjustment
mcuu	Zi	parm_up_micro	Like parm_up_cursor
mcuu1	Zd	micro_up	Like cursor_up for micro adjustment
mgc	MC	clear_margins	Clear all margins top, bottom,
mhpa	ZY	micro_column_address	Like column_address for micro adjustment
mrcup	CM	cursor_mem_address	Memory relative cursor addressing
mvpa	Zc	micro_row_address	Like row_address for micro adjustment
nel	nw	newline	Produces newline (behaves like cr followed by lf)
oc	oc 0	orig_colors	Set all color(pair)s to the original ones
op	op	orig_pair	Set default color pair to the original one
pad	pc	pad_char	Pad character (rather than null)
pfkey	pk	pkey_key	Prog funct key #1 to type string #2
pfloc	pl	pkey_local	Prog funct key #1 to execute string #2
pfx	px	pkey_xmit	Prog funct key #1 to xmit string #2
pln	pn	plab_norm	Prog label #1 to show string #2
porder	Ze	order_of_pins	Matches software bits to print-head pins
prot	mp	enter_protected_mode	Turn on protected mode
rbim	Zs	stop_bit_image	Stop printing bit image graphics
rc	rc	restore_cursor	Restore cursor to position of last sc

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
rcsd	Zt	stop_char_set_def	Stop definition of character set
rep	rp	repeat_char	Repeat char #1 #2 times
rev	mr	enter_reverse_mode	Turn on reverse video mode
rf	rf	reset_file	Name of file containing reset string
rfi	RF	req_for_input	Send next input character (for ptys)
ri	ri	scroll_reverse	Scroll text down
rin	SR	parm_rindex	Scroll backward #1 lines
ritm	ZR	exit_italics_mode	Disable italics
rlm	ZS	exit_leftward_mode	Enable rightward (normal) carriage motion
rmacs	ae	exit_alt_charset_mode	End alternate character set
rmam	RA	exit_am_mode	Turn off automatic margins
rmcup	te	exit_ca_mode	String to begin programs that use cup
rmdc	ed	exit_delete_mode	End delete mode
rmicm	ZT	exit_micro_mode	Disable micro motion capabilities
rmir	ei	exit_insert_mode	End insert mode
rmkx	ke	keypad_local	Out of keypad transmit mode
rmln	LF	label_off	Turn off soft labels
rmm	mo	meta_off	Turn off meta mode
rmp	rP	char_padding	Like ip, but when in replace mode
rmso	se	exit_standout_mode	End stand out mode
rmul	ue	exit_underline_mode	End underscore mode
rmxon	RX	exit_xon_mode	Turn off XON/XOFF handshaking
rs1	r1	reset_1string	Reset terminal completely to sane modes

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
rs2	r2	reset_2string	Reset terminal completely to sane modes
rs3	r3	reset_3string	Reset terminal completely to sane modes
rshm	ZU	exit_shadow_mode	Disable shadow printing
wsubm	ZV	exit_subscript_mode	Disable subscript printing
rsupm	ZW	exit_supercript_mode	Disable superscript printing
rum	ZX	exit_upward_mode	Enable downward (normal) carriage motion
rwidm	ZQ	exit_doublewide_mode	Disable double wide printing
sbim	Zq	start_bit_image	Start printing bit image graphics
sc	sc	save_cursor	Save cursor position
scp	sp	set_color_pair	Set current color pair
scs	Zj	select_char_set	Select character set
scsd	Zr	start_char_set_def	Start definition of character set
sdrfq	ZG	enter_draft_quality	Set draft quality print
setb	Sb	set_background	Set current background color
setf	Sf	set_foreground	Set current foreground color
sgr	sa	set_attributes	Define the video attributes
sgr0	me	exit_attribute_mode	Turn off all attributes
sitm	ZH	enter_italics_mode	Enable italics
slm	ZI	enter_leftward_mode	Enable leftward carriage motion
smacs	as	enter_alt_charset_mode	Start alternate character set
smam	SA	enter_am_mode	Turn on automatic margins
smcup	ti	enter_ca_mode	String to end programs that use cup
smdc	dm	enter_delete_mode	Delete mode (enter)
smgb	Zk	set_bottom_margin	Set bottom margin at current line

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
smgbp	Zl	set_bottom_margin_parm	Set bottom margin at line #1 or #2 from bottom
smgl	ML	set_left_margin	Set left margin at current column
smglp	Zm	set_left_margin_parm	Set left (right) margin at column #1 (#2)
smgr	MR	set_right_margin	Set right margin at current column
smgrp	Zn	set_right_margin_parm	Set right margin at column #1
smgt	Zo	set_top_margin	Set top margin at current line
smgtp	Zp	set_top_margin_parm	Set top (bottom) margin at line #1 (#2)
smicm	ZJ	enter_micro_mode	Enable micro motion capabilities
smir	im	enter_insert_mode	Insert mode (enter)
smln	LO	label_on	Turn on soft labels
smm	mm	meta_on	Turn on meta mode (8th bit)
smso	so	enter_standout_mode	Begin stand out mode
smul	us	enter_underline_mode	Start underscore mode
smxon	SX	enter_xon_mode	Turn on XON/XOFF handshaking
snlq	ZK	enter_near_letter_quality	Set near-letter quality
snrmq	ZL	enter_normal_quality	Set normal quality print
sshm	ZM	enter_shadow_mode	Enable shadow printing
ssubm	ZN	enter_subscript_mode	Enable subscript printing
ssupm	ZO	enter_supercript_mode	Enable superscript printing
subcs	Zu	subscript_characters	List of "subscript-able" characters
sum	ZP	enter_upward_mode	Enable upward carriage msupcs
supcs	Zv	superscript_characters	List of "superscript-able" characters

Table 3-9. String Capabilities (continued)

TInfo	TCap	Variable	Description
`swidm`	`ZF`	`enter_doublewide_mode`	Enable double wide printing
`tbc`	`ct`	`clear_all_tabs`	Clear all tab stops
`tsl`	`ts`	`to_status_line`	Go to status line
`uc`	`uc`	`underline_char`	Underscore one char and move past it
`use`	`tc`	`N/A`	Read capabilities from entry
`vpa`	`cv`	`row_address`	Like **hpa**, but sets row
`wind`	`wi`	`set_window`	Current window is lines #1 - #2 cols #3 - #4
`xoffc`	`XF`	`xoff_character`	XOFF character
`xonc`	`XN`	`xon_character`	XON character
`zerom`	`Zx`	`zero_motion`	No motion for the subsequent character

The following sections group these categories and look at each group in detail.

Configuration Capabilities

Table 3-10 lists the capabilities used to configure terminals. Following the table are more detailed descriptions of some of these capabilities.

Table 3-10. Configuration Capabilities

Capability	Variable	Description
acscU=	acs_chars	Graphics character set pairs aAbBcC-defn=vt100
am	auto_right_margin	Wraps to next line at right margin
bel=	bell	Produce audible signal (bell or beep)
bufsz#	buffer_capacity	Number of bytes buffered before printing
bw	auto_left_margin	cub1 wraps from column 0 to last column
ccc	can_change	Terminal can redefine existing color
chts	hard_cursor	Cursor is hard to see
cmdch=	command_character	Terminal settable command character in prototype
cols#	columns	Number of columns on each line
cr=	carriage_return	Move cursor to left edge of screen
da	memory_above	Display may be retained above the screen
db	memory_below	Display may be retained below the screen
eo	erase_overstrike	Terminal can erase overstrikes with a blank
eslok	status_line_esc_ok	Escape can be used on the status line
gn	generic_type	Generic line type (for example, dialup, switch)
hc	hard_copy	Hardcopy terminal
hs	has_status_line	Terminal has extra status line
hz	tilde_glitch	Hazeltine: cannot print tilde (˜)
if=	init_file	Name of file containing is
iprog=	init_prog	Path name of program for initialization
is1=	init_1string	Terminal initialization string
is2=	init_2string	Terminal initialization string
is3=	init_3string	Terminal initialization string

Table 3-10. Configuration Capabilities (continued)

Capability	Variable	Description
lines#	lines	Number of lines on the screen
lm#	lines_of_memory	Lines of memory if > 0; 0 means unfixed
nxon	needs_xon_xoff	Padding will not work, XON/XOFF required
pad=	pad_char	Pad character (rather than null)
pb#	padding_baud_rate	Lowest baud rate where cr/nl padding needed
os	over_strike	Overstrike leaves both characters on screen
rf=	reset_file	Name of file containing reset string
rs1=	reset_1string	Reset terminal completely to sane modes
rs2=	reset_2string	Reset terminal completely to sane modes
rs3=	reset_3string	Reset terminal completely to sane modes
rmam=	exit_am_mode	Turn off automatic margins
rmxon=	exit_xon_mode	Turn off XON/XOFF handshaking
sam	semi_auto_right_margin	Printing in last column causes cr
smam=	enter_am_mode	Turn on automatic margins
smxon=	enter_xon_mode	Turn on XON/XOFF handshaking
ul	transparent_underline	Underline character overstrikes
vt#	virtual_terminal	Virtual terminal # (CB/UNIX)
xenl	eat_newline_glitch	Newline ignored after 80 columns (Concept)
xoffc=	xoff_character	XOFF character
xon	xon_xoff	Terminal uses XON/XOFF handshaking
xonc=	xon_character	XON character
xsb	no_esc_ctlc	Beehive (f1=escape, f2=ctrl C)
xt	dest_tabs_magic_smso	Tabs destructive, magic smso character

Detailed Descriptions

More detailed descriptions are provided below for the following capabilities:

- cmdch
- da, db
- gn
- if, iprog, is1, is2, is3
- lm
- xenl
- os, hc
- ascs

cmdch. Some terminals, such as the Tektronix 4025, have a control character that can be set. The `cmdch` string describes a "dummy" control character to be used in all capabilities. Some UNIX systems support the convention of using the value of the environment variable `CC` in place of the dummy control character.

da. The `da` capability describes the case where deleting a line or scrolling a full screen may bring non-blank lines from below the screen; the `db` capability describes the case where scrolling back with `ri` may bring down non-blank lines from above.

gn. Terminal descriptions that are not specific types of terminals (such as `switch`, `dialup`, `patch`, and `network`) can be described at a basic level with `gn`. This allows the terminal to function at a low level but still complain when some operations are impossible. This capability is not used for virtual terminal descriptions supported by the UNIX system virtual terminal protocol. (Use `vt`.)

if, iprog, is1, is2, is3. The `if`, `iprog`, `is1`, `is2`, and `is3` strings must be sent to the device every time that the user logs in. They must be output in a certain order: run the program specified by `iprog`; output `is1`; output `is2`; set the margins with `mgc`, `smgl`, and `smgr`; set the tabs with `tbc` and `hts`; print the file specified by `if`; and output `is3` (using the `init` option of the `tput` command).

Normally initialization is done with `is2` and in special cases, `is1` and `is3`; however, sequences that reset from an unknown state can be given as `rs1`, `rs2`, `rf`, and `rs3`. (Reset strings are normally output with the `reset` option of the `tput` command.) Commands are normally placed in these strings when they have annoying results or are not needed when logging in.

lm. The `lm` capability is used if the terminal has more lines of memory than can be displayed on the screen simultaneously; a value of zero means that the number of lines is not fixed but that number is still more than can fit on the screen.

xenl. In addition to terminals that ignore a linefeed, `xenl` should be specified for terminals that do not immediately wrap when a character is read to the right-most column of the screen but wait until another character has been received (the VT100, for example).

os, hc. If the device is a printing terminal with no soft copy unit, both `os` and `hc` should be specified.

ascs. The definition of the `ascs` string is based on the DEC VT100 character set with the addition of some characters from the AT&T 4410v1 terminal. Table 3-11 illustrates the glyph to character mapping.

Table 3-11. Glyph to Character Mapping

Glyph Name	Character
right arrow	+
left arrow	,
down arrow	.
up arrow	-
solid square block	0
lantern symbol	I
diamond	`

Table 3-11. Glyph to Character Mapping (continued)

Glyph Name	Character
checker board (stipple)	`a`
degree symbol	`f`
plus/minus	`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`

Table 3-11. Glyph to Character Mapping (continued)

<table>
<tr><th>Glyph Name</th><th>Character</th></tr>
<tr><td>left tee (|-)</td><td>t</td></tr>
<tr><td>right tee (-|)</td><td>u</td></tr>
<tr><td>bottom tee ()</td><td>v</td></tr>
<tr><td>top tee (T)</td><td>w</td></tr>
<tr><td>vertical line</td><td>x</td></tr>
<tr><td>bullet</td><td>~</td></tr>
</table>

The characters are described in pairs, with the defining character for the glyph followed by the corresponding character on the device. For example, a device with the left tee, right tee, bottom tee, and top tee defined by the `f`,`g`, `h`, and `i` characters would be described as follows:

```
acsc=tfugvhwi
```

Cursor Movement and Scrolling Capabilities

Table 3-12 lists the capabilities used to define cursor movements. Following the table are more detailed descriptions of some of these capabilities.

Table 3-12. Cursor Movement Capabilities

Capability	Variable	Description
`csr=`	`change_scroll_region`	Change to lines #1 through #2 (vt100)
`cub=`	`parm_left_cursor`	Move cursor left specified number of spaces
`cub1=`	`cursor_left`	Move cursor left one space
`cud=`	`parm_down_cursor`	Move down specified number of lines
`cud1=`	`cursor_down`	Move cursor down one line
`cuf=`	`parm_right_cursor`	Move cursor right specified number of spaces
`cuf1=`	`cursor_right`	Non-destructive space (cursor right)
`cup=`	`cursor_address`	Move cursor to row #1 col #2
`cuu=`	`parm_up_cursor`	Move up specified number of lines
`cuu1=`	`cursor_up`	Move cursor up one line
`home=`	`cursor_home`	Move cursor to upper left corner of screen (if no `cup`)
`hpa=`	`column_address`	Absolute horizontal position
`ind=`	`scroll_forward`	Scroll text up
`indn=`	`parm_index`	Scroll forward #1 lines
`ll=`	`cursor_to_ll`	Moves cursor to last line, first column (if no `cup`)
`mir`	`move_insert_mode`	Safe to move in insert mode
`mrcup=`	`cursor_mem_address`	Memory relative cursor addressing
`msgr`	`move_standout_mode`	Safe to move in standout mode
`nel=`	`newline`	Produces newline (behaves like cr followed by lf)
`nrrmc=`	`non_rev_rmcup`	smcup does not restore screen after `rmcup`
`rc=`	`restore_cursor`	Restores cursor to position of last `sc`
`ri=`	`scroll_reverse`	Scrolls text down
`rin=`	`parm_rindex`	Scrolls backward #1 lines

Table 3-12. Cursor Movement Capabilities (continued)

Capability	Variable	Description
`rmcup=`	`exit_ca_mode`	String that ends programs that use `cup`
`sc=`	`save_cursor`	Save cursor position
`smcup=`	`enter_ca_mode`	String that begins programs that use `cup`
`vpa=`	`row_address`	Absolute vertical position
`xhpa`	`col_addr_glitch`	Only positive motion for hpa/mhpa capabilties
`xt`	`dest_tabs_magic_smso`	Tabs destructive, magic `smso` character (Teleray 1061)
`xvpa`	`row_addr_glitch`	Only positive motion for `vpa/mvpa` capabilities

Detailed Descriptions

More detailed descriptions are provided below for the following capabilities:

- ind, ri
- cufl
- csr
- cup, cuu
- home
- mir
- smcup, rmcup
- xt

ind, ri. Local cursor movements defined for `TERMINFO` are undefined at the top and left edges of the screen. Unless `bw` is specified, programs should not try to backspace at the left edge of the screen. To scroll text down, the program should move the cursor to the bottom left corner of the screen and send `ind`. To scroll up, the program should move the cursor to the top of the screen and send `ri`. Both of these capabilities are undefined when the cursor is anywhere else on the screen.

Although moving backwards from the left edge of the screen is not possible unless `bw` is specified, moving forward at the right edge of the screen does not necessarily depend on whether or not `am` is set. If the device has switch selectable automatic margins, `am` should be specified in the `TERMINFO` file, and initialization strings should turn on this option.

cufl. The local cursor movements should not change the text that they pass over; for example, you would not use `cufl=\s` because the space would erase the character that it passed over.

csr. The cursor position is undefined after using `csr`. Do not specify `csr` on terminals that do not have destructive scroll regions unless all of the following simulate destructive scrolling: `ind`, `ri`, `indn`, `rin`, `dl`, and `dl1`. To find out whether or not the terminal has a destructive scroll region, create a scroll region and place data on the bottom line, move the cursor to the top and do a reverse index (`ri`), followed by a delete line (`dl1`) or index (`ind`). The terminal has destructive scroll regions if the bottom line drops off the bottom of the scrolling region.

The `csr` string can create the effect of insert and delete line.

cup, cuu. If the device does not have `cup`, (Tektronix 4025, for example), `cud`, `cub`, `cuf`, and cuu are useful for moving relative to present position. Sometimes absolute-cursor addressing in one dimension (using `vpa` and `hpa`) is faster than the two-parameter specification of `cup` (as with the Hewlett-Packard 2645).

home. Since `home` refers to the top left corner of the screen (0,0) and not to memory, the sequence `\EH` on Hewlett-Packard terminals could not be used without losing other features of the terminal. A program should not get to the lower left hand corner by going up with `cuu1` itself from the home position because the effects of `home` cannot be predicted.

ri, ind. The `ri` and `ind` strings can be used to insert lines at the top or bottom of the screen on terminals that have no true insert/delete line (and may even be faster).

mir. A terminal that allows movement within insert mode to delete characters on the same line can specify `mir` to speed up the insertion process. Some terminals, such as Datamedia's, should not specify `mir` because of the way that their insert mode works.

smcup, rmcup. Some devices may need to be in a special mode to use cursor movement. Some terminals turn off cursor movement when not in use; other terminals, such as the Concept terminal, have cursor addressing relative to memory instead of the screen; so a single screen-sized window must be fixed into the device for cursor addressing to work properly. Terminals, such as the Tektronix 4025, that have programmable command characters need to set the command character to the one used in `TERMINFO`. The `smcup` and `rmcup` strings start and end programs that use cursor movement.

xt. The `xt` capability is used for Teleray terminals that have destructive tabs (turn all characters tabbed over to blanks) and have an odd standout mode that requires using insert and delete to change text from standout back to normal instead of typing over the text.

Edit Capabilities

Table 3-13 lists the capabilities used for inserting and deleting text on terminals. Following the table are more detailed descriptions of the following capabilities:

- ich1
- in
- ip

Table 3-13. Editing Capabilities

Capability	Variable	Description
clear=	clear_screen	Clear screen
dch=	parm_dch	Delete #1 chars
dch1=	delete_character	Delete character
dl=	parm_delete_line	Delete #1 lines (only from first column on line)
dl1=	delete_line	Delete current line (only from first position on line)
ech=	erase_ch	Clear from current position to end of screen
el=	clr_eol	Clear from current position to end of line, leaving the cursor in the original position
el1=	clr_bol	Clear from current position to beginning of line inclusive, leaving the cursor in the original position
ich=	parm_ich	Insert #1 blank characters
ich1=	insert_character	Insert character
il=	parm_insert_line	Add #1 new blank lines (only from first column line)
il1=	insert_line	Add new blank line above cursor (only from first column on line)
in	insert_null_glitch	Insert mode distinguishes nulls

Table 3-13. Editing Capabilities (continued)

Capability	Variable	Description
`ip=`	`insert_padding`	Insert pad after character inserted
`rmdc=`	`exit_delete_mode`	End delete mode
`rmir=`	`exit_insert_mode`	End insert mode
`rmp=`	`char_padding`	Like `ip`, but when in replace mode
`smdc=`	`enter_delete_mode`	Delete mode (enter)
`smir=`	`enter_insert_mode`	Insert mode (enter)

Detailed Descriptions

More detailed descriptions are provided below for the following capabilities:

- ich1
- in
- ip

ich1. Most characters with a true insert mode do not require `ich1` to be specified; specify `ich1` for any terminal that requires a sequence to be sent before a character can be inserted. (Do not specify both `smir` and `ich1` unless the terminal must be placed in insert mode and must be sent a character before insertion).

in. If the terminal distinguishes between blank characters and untyped positions (for example, local cursor movements) `in` should be specified. To find out, clear the screen, type a few characters, press a cursor key, and type a few more characters. Put the cursor before the first set of characters and put the terminal in insert mode. If the first set of characters shifts to the second and then moves onto the next line as you insert more characters, the terminal should be described by `in`.

ip. In addition to post-insert padding, `ip` can be used for any sequence that has to be sent after a character is inserted.

Attribute Capabilities

Table 3-14 lists the capabilities used to control attributes on the terminal. For example, attribute capabilities include those to manipulate bold, underline, and color.

Table 3-14. Attribute Capabilities

Capability	Variable	Description
`bce`	`back_color_erase`	Screen erased with background color
`blink=`	`enter_blink_mode`	Turn on blinking
`bold=`	`enter_bold_mode`	Turn on bold (extra bright) mode
`colors#`	`max_colors`	Maximum number of colors on the screen
`dim=`	`enter_dim_mode`	Turn on half-bright mode
`hls`	`hue_lightness_saturation`	Terminal uses only HLS color notation (Tektronix)
`initc=`	`initialize_color`	Initialize the definition of color
`initp=`	`initialize_pair`	Initialize color pair
`invis=`	`enter_secure_mode`	Turn on blank mode (characters invisible)
`ncv#`	`no_color_video`	Video attributes that cannot be used with colors
`oc=`	`orig_colors`	Set all color (pair)s to the original one
`op=`	`orig_pair`	Set default color pair to the original one
`pairs#`	`max_pairs`	Maximum number of color pairs on the screen
`prot=`	`enter_protected_mode`	Turn on protected mode
`rev=`	`enter_reverse_mode`	Turn on reverse video mode
`rmso=`	`exit_standout_mode`	End stand out mode
`rmul=`	`exit_underline_mode`	End underscore mode
`scp=`	`set_color_pair`	Set current color pair

Table 3-14. Attribute Capabilities (continued)

Capability	Variable	Description
setb=	set_background	Set current background color
setf=	set_foreground	Set current foreground color
sgr=	set_attributes	Define the video attributes
sgr0=	exit_attribute_mode	Turn off all attributes
smso=	enter_standout_mode	Begin stand out mode
smul=	enter_underline_mode	Start underscore mode
xhp	ceol_standout_glitch	Standout not erased by overwriting (hp)
xmc#	magic_cookie_glitch	# of blank chars left by smso or rmso

Handling Color

There are two ways terminals handle color. The Tektronix method provides a fixed set of colors that can be used for background or foreground. The Hewlett-Packard method provides color pairs that represent the foreground and background colors together; there is no way to define the foreground and background independent of each other.

The `initc` string is defined for terminals that use the Tektronix method of handling color. The `initc` string requires four arguments: one color number (0 to `color -1`), and three RGB (red, green, blue) values or three HLS colors (hue, lightness, saturation) in the same order specified in the parentheses.

The `initp` string is defined for terminals that use the Hewlett-Packard method of handling color. The `initp` string requires seven parameters: the number of the color pair (0 to `pairs -1`), three RGB values for foreground, and three RGB values for background (each group in the order of red, green, blue).

The `hls` Boolean variable is used to tell the `CURSES init_color( )` routine to convert RGB (red, green, blue) arguments to HLS (hue, lighness, saturation) before sending them to the terminal (for those terminals that only use HLS notation).

Some color terminals replace video attributes with colors. Since these attributes should not be combined with colors, they need to be identified. Information about these attributes are packed into the `ncv` variable.

ncv Variable. The nine least significant bits of the `ncv` variable correspond to the video attributes as shown in Table 3-15.

Table 3-15. ncv Variable

Attribute	Bit Position	Decimal Value
`A_STANDOUT`	0	1
`A_UNDERLINE`	1	2
`A_REVERSE`	2	4
`A_BLINK`	3	8
`A_DIM`	4	16
`A_BOLD`	5	32
`A_INVIS`	6	64
`A_PROTECT`	7	128
`A_ALTCHARSET`	8	256

The corresponding `ncv` bit of each attribute that should not be combined with color should be set to one; otherwise, it should be set to zero. The decimal values of the attributes that cannot be used with color are summed, and that sum is packed into the `ncv` variable. For example, if the terminal uses color for standout mode (decimal value 1) and for underlining (decimal value 2), `ncv` would be a value of 3.

Turning Off Attributes

Always use `sgr0` to turn off video attributes since it is the only way to turn off some attributes like `dim` or `blink`.

Programs using standout mode should exit standout mode (`rmso`) before sending a newline or moving the cursor unless the `msgr` capability is present. (The `msgr` capability specifies that it's safe to move in standout mode.)

Setting Arbitrary Modes

The `sgr` string describes the sequence to set arbitrary combinations of modes. The `sgr` string takes nine parameters in the following order:

1. standout
2. underline
3. reverse
4. blink
5. dim
6. bold
7. blank
8. protect
9. alternate character set

Each parameter is either zero or non-zero, representing whether the attribute is on or off.

Table 3-16 lists the `sgr` parameters.

Table 3-16. sgr Parameters

tparm	**Attribute**
	none
`p1`	standout
`p2`	underline
`p3`	reverse
`p4`	blink
`p5`	dim
`p6`	bold
`p7`	invis
`p8`	protect
`p9`	altcharset

Tabs and Margins

Table 3-17 lists the capabilities used to control margins and tabs on the terminal.

Table 3-17. Margins and Tabs

Capability	Variable	Description
`cbt=`	`back_tab`	Back tab
`ht=`	`tab`	Tab to next eight space hardware tab stop
`hts=`	`set_tab`	Set a tab in all rows, current column
`it#`	`init_tabs`	Number of spaces between initial tabs
`mgc=`	`clear_margins`	Clear all margins top, bottom,
`smgb=`	`set_bottom_margin`	Set bottom margin at current line
`smgbp=`	`set_bottom_margin_parm`	Set bottom margin at line #1 or #2 from bottom
`smgl=`	`set_left_margin`	Set left margin at current column
`smglp=`	`set_left_margin_parm`	Set left (right) margin at column #1 (#2)
`smgr=`	`set_right_margin`	Set right margin at current column
`smgrp=`	`set_right_margin_parm`	Set right margin at column #1
`smgt=`	`set_top_margin`	Set top margin at current line
`smgtp=`	`set_top_margin_parm`	Set top (bottom) margin at line #1 (#2)
`tbc=`	`clear_all_tabs`	Clear all tab stops

If tabs are expanded by the computer rather than sent to the device, by convention, `ht` and `cbt` are not used because the user may not have the tabs correctly set.

If the sequence to set tabs cannot be described adequately with `tbc` and `hts`, it can be described with `is2` or `if`.

The `it` string is normally used by `tputs init` to determine whether to set the mode for hardware expansion and whether to set the tabs.

Terminal Key Capabilities

Table 3-18 lists the capabilities used to describe keys on the terminal.

Table 3-18. Terminal Key Capabilities

Capability	Variable	Description
`k_keyname=`	See Table 3-9	See Table 3-9
`km`	`has_meta_key`	Terminal has meta key (shift, sets parity bit)
`lab_function_name=`	See Table 3-9	See Table 3-9
`lf0=`	`lab_f0`	Labels on function key f0 if not f0
`lf1=`	`lab_f1`	Labels on function key f1 if not f1
`lf2=`	`lab_f2`	Labels on function key f2 if not f2
`lf3=`	`lab_f3`	Labels on function key f3 if not f3
`lf4=`	`lab_f4`	Labels on function key f4 if not f4
`lf5=`	`lab_f5`	Labels on function key f5 if not f5
`lf6=`	`lab_f6`	Labels on function key f6 if not f6
`lf7=`	`lab_f7`	Labels on function key f7 if not f7
`lf8=`	`lab_f8`	Labels on function key f8 if not f8
`lf9=`	`lab_f9`	Labels on function key f9 if not f9
`lf10=`	`lab_f10`	Labels on function key f10 if not f10
`pfkey=`	`pkey_key`	Prog funct key #1 to type string #2
`pfloc=`	`pkey_local`	Prog funct key #1 to execute string #2
`pfx=`	`pkey_xmit`	Prog funct key #1 to xmit string #2
`rfi=`	`req_for_input`	Send next input character (for ptys)
`rmkx=`	`meta_off`	Turn off meta mode
`smkx=`	`keypad_xmit`	Put terminal in keypad transmit mode
`smm=`	`meta_on`	Turn on meta mode (8th bit)

Note Refer back to Table 3-9 for a complete list of all the key capabilities.

Key capabilities describe keypads that transmit sequences of characters when keys are pressed. Keypads that work only in local mode cannot be described. The keypad is assumed to always transmit; if the transmit of keys can be turned on or off, this should be specified with `smkx` and `rmkx`, respectively.

If the first 11 function keys have labels other than the default `f0` through `f10`, they can be described using `lf0` through `lf10`. If the keypad has a `3` by `3` array of keys that includes the four arrow keys, the other keys can be described as `ka1`, `ka3`, `kb2`, `kc1`, and `kc3` as shown in the diagram below.

`ka1`	▲	ka3
◀	kb2	▶
`kc1`	▼	kc3

Miscellaneous Capabilities

Table 3-19 lists the capabilities that do not fit into any of the previous categories.

Table 3-19. Miscellaneous Capabilities

Capability	Variable	Description
`civis=`	`cursor_invisible`	Make cursor invisible
`cnorm=`	`cursor_normal`	Make cursor appear normal (undo `cvvis/civis`)
`cvvis=`	`cursor_visible`	Make cursor very visible
`defc=`	`define_char`	Define a character in a character set
`dsl=`	`dis_status_line`	Disable status line
`enacs=`	`ena_acs`	Enable alternate character set

Table 3-19. Miscellaneous Capabilities (continued)

Capability	Variable	Description
`flash=`	`flash_screen`	Visible bell (may not move cursor)
`ff=`	`form_feed`	Hardcopy terminal page eject
`fsl=`	`from_status_line`	Return from status line
`lh#`	`label_height`	# rows in each label
`lw#`	`label_width`	# columns in each label
`nlab#`	`num_labels`	# of labels on screen (start at 1)
`pln=`	`plab_norm`	Prog label #1 to show string #2
`rbim=`	`stop_bit_image`	Stop printing bit image graphics
`rcsd=`	`stop_char_set_def`	Stop definition of character set
`rep=`	`repeat_char`	Repeat character #1 #2 times
`rmacs=`	`exit_alt_charset_mode`	End alternate character set
`rmln=`	`label_off`	Turn off soft labels
`sbim=`	`start_bit_image`	Start printing bit image graphics
`scs=`	`select_char_set`	Select character set
`scsd=`	`start_char_set_def`	Start definition of character set
`smacs=`	`enter_alt_charset_mode`	Start alternate character set
`smln=`	`label_on`	Turn on soft labels
`tsl=`	`to_status_line`	Go to status line
`use=`	`N/A`	Read capabilities from entry
`wind=`	`set_window`	Current window is lines #1 - #2 cols #3 - #4
`wsl#`	`width_status_line`	# columns in status line (if different from *cols*)

The `fsl` string must leave the cursor in the same position as it was before `tsl`. If necessary, this can be done by including `sc` and `rc` in the`fsl` and `tsl` strings.

The `wind` string defines a window that all commands affect as part of memory. It takes four arguments: starting lines in memory, ending lines in memory, starting columns in memory, and ending columns in memory.

Capabilities Sorted by Variable Name

The following tables list all of the capabilities by variable name (instead of `TERMINFO` capability name as was done previously) for ease of reference.

Table 3-20 lists the Boolean capabilities, Table 3-21 lists the numeric capabilities, and Table 3-22 lists the string capabilities.

Boolean Capabilities

Table 3-20 lists the Boolean capabilities.

Table 3-20. Boolean Capabilities

Variable	TInfo	TCap	Description
auto_left_margin	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
back_color_erase	bce	be	Screen erased with background color
can_change	ccc	cc	Terminal can redefine existing color
ceol_standout_glitch	xhp	xs	Standout not erased by overwriting (hp)
col_addr_glitch	xhpa	YA	Only positive motion for hpa/mhpa capabilities
cpi_changes_res	cpix	YF	Changing character pitch changes resolution
cr_cancels_micro_mode	crxm	YB	Using cr turns off micro mode
dest_tabs_magic_smso	xt	xt	Tabs destructive, magic smso character (Teleray 1061)
eat_newline_glitch	xenl	xn	Newline ignored after 80 columns (Concept)
erase_overstrike	eo	eo	Terminal can erase overstrikes with a blank
generic_type	gn	gn	Generic line type (for example, dialup, switch)
hard_copy	hc	hc	Hardcopy terminal
hard_cursor	chts	HC	Cursor is hard to see
has_meta_key	km	km	Terminal has meta key (shift, sets parity bit)
has_print_wheel	daisy	YC	Printer needs operator to change character set
has_status_line	hs	hs	Terminal has extra status line
hue_lightness_saturation	hls	hl	Terminal uses only HLS color notation (Tektronix)

Table 3-20. Boolean Capabilities (continued)

Variable	TInfo	TCap	Description
`insert_null_glitch`	`in`	`in`	Insert mode distinguishes nulls
`lpi_changes_res`	`lpix`	`YG`	Changing line pitch changes resolution
`memory_above`	`da`	`da`	Display may be retained
`amemory_below`	`db`	`db`	Display may be retained below the screen
`move_insert_mode`	`mir`	`mi`	Safe to move in insert mode
`move_standout_mode`	`msgr`	`ms`	Safe to move in standout mode
`needs_xon_xoff`	`nxon`	`nx`	Padding will not work, XON/XOFF required
`no_esc_ctlc`	`xsb`	`xb`	Beehive (f1=escape, f2=ctrl C)
`no_pad_char`	`npc`	`NP`	Pad character does not exist
`non_rev_rmcup`	`nrrmc`	`NR`	`smcup` does not reverse `rmcup`
`over_strike`	`os`	`os`	Terminal overstrikes
`prtr_silent`	`mc5i`	`5i`	Printer will not echo on screen
`row_addr_glitch`	`xvpa`	`YD`	Only positive motion for vpa/mvpa capabilities
`semi_auto_right_margin`	`sam`	`YE`	Printing in last column causes cr
`status_line_esc_ok`	`eslok`	`es`	Escape can be used on the status line
`tilde_glitch`	`hz`	`hz`	Hazeltine: cannot print tilde (~)
`transparent_underline`	`ul`	`ul`	Underline character overstrikes
`xon_xoff`	`xon`	`xo`	Terminal uses XON/XOFF handshaking

Numeric Capabilities

Table 3-21 lists the numeric capabilities.

Table 3-21. Numeric Capabilities

Variable	TInfo	TCap	Description
`buffer_capacity`	`bufsz`	`Ya`	Number of bytes buffered before printing
`columns`	`cols`	`co`	Number of columns in a line
`dot_horz_spacing`	`spinh`	`Yc`	Spacing of dots horizontally in dots per inch
`dot_vert_spacing`	`spinv`	`Yb`	Spacing of pins vertically in pins per inch
`init_tabs`	`it`	`it`	Number of spaces between initial tabs
`label_height`	`lh`	`lh`	Number of rows in each label
`label_width`	`lw`	`lw`	Number of columns in each label
`lines`	`lines`	`li`	Number of lines on screen or page

Table 3-21. Numeric Capabilities (continued)

Variable	TInfo	TCap	Description
lines_of_memory	lm	lm	Lines of memory if > 0; 0 means unfixed
magic_cookie_glitch	xmc	sg	Number of blank chars left by smso or rmso
max_colors	colors	Co	Maximum number of colors on the screen
max_micro_address	maddr	Yd	Maximum value in micro_ ... _address
max_micro_jump	mjump	Ye	Maximum value in parm_ ... _micro
max_pairs	pairs	pa	Maximum number of color pairs on the screen
micro_col_size	mcs	Yf	Character step size when in micro mode
micro_line_size	mls	Yg	Line step size when in micro mode
no_color_video	ncv	NC	Video attributes that cannot be used with colors
num_labels	nlab	Nl	Number of labels on screen (start at 1)
number_of_pins	npins	yH	Number of pins in print-head
output_res_char	orc	Yi	Horz. res. in units per character
output_res_horz_inch	orhi	Yk	Horz. res. in units per inch
output_res_line	orl	Yj	Vert. res. in units per line
output_res_vert_inch	orvi	Yl	Vert. res. in units per inch
padding_baud_rate	pb	pb	Lowest baud rate where cr/nl padding needed
virtual_terminal	vt	vt	Virtual terminal number (CB/UNIX)
wide_char_size	widcs	Yn	Character step size when in double wide mode
width_status_line	wsl	ws	Number of columns in status line

String Capabilities

Table 3-22 lists the string capabilities.

Table 3-22. String Capabilities

Variable	TInfo	TCap	Description
N/A	use	tc	Read capabilities from entry
acs_chars	acsc	ac	Graphics character set pairs aAbBcC - defn=vt100
back_tab	cbt	bt	Back tab
bell	bel	bl	Produce audible signal (bell or beep)
carriage_return	cr	cr	Move cursor to left edge of screen
change_char_pitch	cpi	ZA	Change number of characters per inch
change_line_pitch	lpi	ZB	Change number of lines per inch
change_res_horz	chr	ZC	Change horizontal resolution
change_res_vert	cvr	ZD	Change vertical resolution
change_scroll_region	csr	cs	Change to lines #1 through #2 (vt100)
char_padding	rmp	rP	Like ip, but when in replace mode
char_set_names	csnm	Zy	List of character set names
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear all margins top and bottom
clear_screen	clear	cl	Clear screen
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display
column_address	hpa	ch	Set cursor column
command_character	cmdch	CC	Terminal settable command character in prototype
cursor_address	cup	cm	Cursor motion to row #1 col #2

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
cursor_down	cud1	do	Move cursor down one line
cursor_home	home	ho	Home cursor (if no cup)
cursor_invisible	civis	vi	Make cursor invisible
cursor_left	cub1	le	Move cursor left one space
cursor_mem_address	mrcup	CM	Memory relative cursor addressing
cursor_normal	cnorm	ve	Make cursor appear normal (undo cvvis/civis)
cursor_right	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll	ll	ll	Last line, first column (if no cup)
cursor_up	cuu1	up	Move cursor up
cursor_visible	cvvis	vs	Make cursor very visible
define_char	defc	ZE	Define a character in a character set
delete_character	dch1	dc	Delete character
delete_line	dl1	dl	Delete line
dis_status_line	dsl	ds	Disable status line
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate character set
enter_alt_charset_mode	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins
enter_blink_mode	blink	mb	Turn on blinking
enter_bold_mode	bold	md	Turn on bold (extra bright) mode
enter_ca_mode	smcup	ti	String to end programs that use cup
enter_delete_mode	smdc	dm	Delete mode (enter)
enter_dim_mode	dim	mh	Turn on half-bright mode

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
enter_doublewide_mode	swidm	ZF	Enable double wide printing
enter_draft_quality	sdrfq	ZG	Set draft quality print
enter_insert_mode	smir	im	Insert mode (enter)
enter_italics_mode	sitm	ZH	Enable italics
enter_leftward_mode	slm	ZI	Enable leftward carriage motion
enter_micro_mode	smicm	ZJ	Enable micro motion capabilities
enter_near_letter_quality	snlq	ZK	Set near-letter quality
enter_normal_quality	snrmq	ZL	Set normal quality print
enter_protected_mode	prot	mp	Turn on protected mode
enter_reverse_mode	rev	mr	Turn on reverse video mode
enter_secure_mode	invis	mk	Turn on blank mode (characters invisible)
enter_shadow_mode	sshm	ZM	Enable shadow printing
enter_standout_mode	smso	so	Begin stand out mode
enter_subscript_mode	ssubm	ZN	Enable subscript printing
enter_supercript_mode	ssupm	ZO	Enable superscript printing
enter_underline_mode	smul	us	Start 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	Erase #1 characters
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
exit_attribute_mode	sgr0	me	Turn off all attributes
exit_ca_mode	rmcup	te	String to begin programs that use cup

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
exit_delete_mode	rmdc	ed	End delete mode
exit_doublewide_mode	rwidm	ZQ	Disable double wide printing
exit_insert_mode	rmir	ei	End 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	End stand out mode
exit_subscript_mode	wsubm	ZV	Disable subscript printing
exit_supercript_mode	rsupm	ZW	Disable superscript printing
exit_underline_mode	rmul	ue	End underscore mode
exit_xon_mode	rmxon	RX	Turn off XON/XOFF handshaking
flash_screen	flash	vb	Visible bell (may not move cursor)
form_feed	ff	ff	Hardcopy terminal page eject
from_status_line	fsl	fs	Return from status line
init_1string	is1=	i1	Terminal initialization string
init_2string	is2=	is	Terminal initialization string
init_3string	is3=	i3	Terminal initialization string
init_file	if	if	Name of file containing **is**
init_prog	iprog	iP	Pathname of program for initialization
initialize_color	initc	Ic	Initialize the definition of color
initialize_pair	initp	Ip	Initialize color pair
insert_character	ich1	ic	Insert character

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
`insert_line`	`il1`	`al`	Add new blank line
`insert_padding`	`ip`	`ip`	Insert pad after character inserted
`key_a1`	`ka1`	`K1`	Upper left of keypad
`key_a3`	`ka3`	`K3`	Upper right of keypad
`key_b2`	`kb2`	`K2`	Center of keypad
`key_backspace`	`kbs`	`kb`	Sent by backspace key
`key_beg`	`kbeg`	`@1`	Sent by beginning key
`key_btab`	`kcbt`	`kB`	Sent by BackTab key
`key_c1`	`kc1`	`K4`	Lower left of keypad
`key_c3`	`kc3`	`K5`	Lower right of keypad
`key_cancel`	`kcan`	`@2`	Sent by cancel key
`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`	Sent by close key
`key_command`	`kcmd`	`@4`	Sent by command key
`key_copy`	`kcpy`	`@5`	Sent by copy key
`key_create`	`kcreate`	`@6`	Sent by create key
`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_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`	Sent by end key

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
key_enter	kent	@8	Sent by enter/send key
key_eol	kel	kE	Sent by clear-to-end-of-line key
key_eos	ked	kS	Sent by clear-to-end-of-screen key
key_exit	kext	@9	Sent by exit key
key_f0	kf0	k0	Sent by function key f0
key_f1	kf1	k1	Sent by function key f1
key_f2	kf2	k2	Sent by function key f2
key_f3	kf3	k3	Sent by function key f3
key_f4	kf4	k4	Sent by function key f4
key_f5	kf5	k5	Sent by function key f5
key_f6	kf6	k6	Sent by function key f6
key_f7	kf7	k7	Sent by function key f7
key_f8	kf8	k8	Sent by function key f8
key_f9	kf9	k9	Sent by function key f9
key_f10	kf10	k;	Sent by function key f10
key_f11	kf11	F1	Sent by function key 11
key_f12	kf12	F2	Sent by function key 12
key_f13	kf13	F3	Sent by function key 13
key_f14	kf14	F4	Sent by function key 14
key_f15	kf15	F5	Sent by function key 15
key_f16	kf16	F6	Sent by function key 16
key_f17	kf17	F7	Sent by function key 17
key_f18	kf18	F8	Sent by function key 18

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
key_f19	kf19	F9	Sent by function key 19
key_f20	kf20	FA	Sent by function key 20
key_f21	kf21	FB	Sent by function key 21
key_f22	kf22	FC	Sent by function key 22
key_f23	kf23	FD	Sent by function key 23
key_f24	kf24	FE	Sent by function key 24
key_f25	kf25	FF	Sent by function key 25
key_f26	kf26	FG	Sent by function key 26
key_f27	kf27	FH	Sent by function key 27
key_f28	kf28	FI	Sent by function key 28
key_f29	kf29	FJ	Sent by function key 29
key_f30	kf30	FK	Sent by function key 30
key_f31	kf31	FL	Sent by function key 31
key_f32	kf32	FM	Sent by function key 32
key_f33	kf33	FN	Sent by function key 33
key_f34	kf34	FO	Sent by function key 34
key_f35	kf35	FP	Sent by function key 35
key_f36	kf36	FQ	Sent by function key 36
key_f37	kf37	FR	Sent by function key 37
key_f38	kf38	FS	Sent by function key 38
key_f39	kf39	FT	Sent by function key 39
key_f40	kf40	FU	Sent by function key 40
key_f41	kf41	FV	Sent by function key 41

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
key_f42	kf42	FW	Sent by function key 42
key_f43	kf43	FX	Sent by function key 43
key_f44	kf44	FY	Sent by function key 44
key_f45	kf45	FZ	Sent by function key 45
key_f46	kf46	Fa	Sent by function key 46
key_f47	kf47	Fb	Sent by function key 47
key_f48	kf48	Fc	Sent by function key 48
key_f49	kf49	Fd	Sent by function key 49
key_f50	kf50	Fe	Sent by function key 50
key_f51	kf51	Ff	Sent by function key 51
key_f52	kf52	Fg	Sent by function key 52
key_f53	kf53	Fh	Sent by function key 53
key_f54	kf54	Fi	Sent by function key 54
key_f55	kf55	Fj	Sent by function key 55
key_f56	kf56	Fk	Sent by function key 56
key_f57	kf57	Fl	Sent by function key 57
key_f58	kf58	Fm	Sent by function key 58
key_f59	kf59	Fn	Sent by function key 59
key_f60	kf60	Fo	Sent by function key 60
key_f61	kf61	Fp	Sent by function key 61
key_f62	kf62	Fq	Sent by function key 62
key_f63	kf63	Fr	Sent by function key 63
key_find	kfnd	@0	Sent by find key

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
key_help	khlp	%1	Sent by help key
key_home	khome	kh	Sent by home key
key_ic	kich1	kI	Sent by ins char/enter ins mode key
key_il	kil1	kA	Sent by insert line
key_left	kcub1	kl	Sent by terminal left-arrow key
key_ll	kll	kH	Sent by home-down key (lower left)
key_mark	kmrk	%2	Sent by mark key
key_message	kmsg	%3	Sent by message key
key_move	kmov	%4	Sent by move key
key_next	knxt	%5	Sent by next-object key
key_npage	knp	kN	Sent by next-page key
key_open	kopn	%6	Sent by open key
key_options	kopt	%7	Sent by options key
key_ppage	kpp	kP	Sent by previous-page key
key_previous	kprv	%8	Sent by previous-object key
key_print	kprt	%9	Sent by print or copy key
key_redo	krdo	%0	Sent by redo key
key_reference	kref	&1	Sent by reference key
key_refresh	krfr	&2	Sent by refresh key
key_replace	krpl	&3	Sent by replace key
key_restart	krst	&4	Sent by restart key
key_resume	kres	&5	Sent by resume key
key_right	kcuf1	kr	Sent by terminal right arrow key

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
key_save	ksav	&6	Sent by save key
key_sbeg	kBEG	&9	Sent by shift-beginning key
key_scancel	kCAN	&0	Sent by shift-cancel key
key_scommand	kCMD	*1	Sent by shift-command key
key_scopy	kCPY	*2	Sent by shift-copy key
key_screate	kCRT	*3	Sent by shift-create key
key_sdc	kDC	*4	Sent by shift-delete-char key
key_sdl	kDL	*5	Sent by shift-delete-line key
key_select	kslt	*6	Sent by select key
key_send	kEND	*7	Sent by shift-end key
key_seol	kEOL	*8	Sent by shift-eol key
key_sexit	kEXT	*9	Sent by shift-exit key
key_sf	kind	kF	Sent by scroll-forward/down key
key_sfind	kFND	*0	Sent by shift-find key
key_shelp	kHLP	#1	Sent by shift-help key
key_shome	kHOM	#2	Sent by shift-home key
key_sic	kIC	#3	Sent by shift-insert-char key
key_sleft	kLFT	#4	Sent by shift-left key
key_smessage	kMSG	%a	Sent by shift-message key
key_smove	kMOV	%b	Sent by shift-move key
key_snext	kNXT	%c	Sent by shift-next key
key_soptions	kOPT	%d	Sent by shift-options key
key_sprevious	kPRV	%e	Sent by shift-prev key

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
key_sprint	kPRT	%f	Sent by shift-print key
key_sr	kri	kR	Sent by scroll-backward/up key
key_sredo	kRDO	%g	Sent by shift-redo key
key_sreplace	kRPL	%h	Sent by shift-replace key
key_sright	kRIT	%i	Sent by shift-right key
key_srsume	kRES	%j	Sent by shift-resume key
key_ssave	kSAV	!1	Sent by shift-save key
key_ssuspend	kSPD	!2	Sent by shift-suspend key
key_stab	khts	kT	Sent by set-tab key
key_sundo	kUND	!3	Sent by shift-undo key
key_suspend	kspd	&7	Sent by suspend key
key_undo	kund	&8	Sent by undo key
key_up	kcuu1	ku	Sent by terminal up-arrow key
keypad_local	rmkx	ke	Out of keypad transmit mode
keypad_xmit	smkx	ks	Put terminal in keypad transmit mode
lab_f0	lf0	l0	Labels on function key f0 if not f0
lab_f1	lf1	l1	Labels on function key f1 if not f1
lab_f10	lf10	la	Labels on function key f10 if not f10
lab_f2	lf2	l2	Labels on function key f2 if not f2
lab_f3	lf3	l3	Labels on function key f3 if not f3
lab_f4	lf4	l4	Labels on function key f4 if not f4
lab_f5	lf5	l5	Labels on function key f5 if not f5
lab_f6	lf6	l6	Labels on function key f6 if not f6

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
lab_f7	lf7	l7	Labels on function key f7 if not f7
lab_f8	lf8	l8	Labels on function key f8 if not f8
lab_f9	lf9	l9	Labels on function key f9 if not f9
label_off	rmln	LF	Turn off soft labels
label_on	smln	LO	Turn on soft labels
meta_off	rmm	mo	Turn off meta mode
meta_on	smm	mm	Turn on meta mode (8th bit)
micro_column_address	mhpa	ZY	Like column_address for micro adjustment
micro_down	mcud1	ZZ	Like cursor_down for micro adjustment
micro_left	mcub1	Za	Like cursor_left for micro adjustment
micro_right	mcuf1	Zb	Like cursor_right for micro adjustment
micro_row_address	mvpa	Zc	Like row_address for micro adjustment
micro_up	mcuu1	Zd	Like cursor_up for micro adjustment
newline	nel	nw	Produces newline (behaves like cr followed by lf)
order_of_pins	porder	Ze	Matches software bits to print-head pins
orig_colors	oc	oc	Set all color(pair)s to the original ones
orig_pair	op	op	Set default color pair to the original one
pad_char	pad	pc	Pad character (rather than null)
parm_dch	dch	DC	Delete #1 characters
parm_delete_line	dl	DL	Delete #1 lines
parm_down_cursor	cud	DO	Move cursor down #1 lines
parm_down_micro	mcud	Zf	Like parm_down_cursor
parm_ich	ich	IC	Insert #1 blank characters

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
parm_index	indn	SF	Scroll forward #1 lines
parm_insert_line	il	AL	Add #1 new blank lines
parm_left_cursor	cub	LE	Move cursor left #1 spaces
parm_left_micro	mcub	Zg	Like parm_left_cursor
parm_right_cursor	cuf	RI	Move cursor right #1 spaces
parm_right_micro	mcuf	Zh	Like parm_right_cursor
parm_rindex	rin	SR	Scroll backward #1 lines
parm_up_cursor	cuu	UP	Move cursor up #1 lines
parm_up_micro	mcuu	Zi	Like parm_up_cursor
pkey_key	pfkey	pk	Prog funct key #1 to type string #2
pkey_local	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit	pfx	px	Prog funct key #1 to xmit string #2
plab_norm	pln	pn	Prog label #1 to show string #2
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	p0	Turn on the printer for #1 bytes
prtr_off	mc4	pf	Turn off the printer
prtr_on	mc5	po	Turn on the printer
repeat_char	rep	rp	Repeat char #1 #2 times
req_for_input	rfi	RF	Send next input character (for ptys)
reset_1string	rs1	r1	Reset terminal completely to sane modes
reset_2string	rs2	r2	Reset terminal completely to sane modes
reset_3string	rs3	r3	Reset terminal completely to sane modes
reset_file	rf	rf	Name of file containing reset string

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
restore_cursor	rc	rc	Restore cursor to position of last sc
row_address	vpa	cv	Like hpa but sets row
save_cursor	sc	sc	Save cursor position
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	ri	Scroll text down
select_char_set	scs	Zj	Select character set
set_attributes	sgr	sa	Define the video attributes
set_background	setb	Sb	Set current background color
set_bottom_margin	smgb	Zk	Set bottom margin at current line
set_bottom_margin_parm	smgbp	Zl	Set bottom margin at line #1 or #2 from bottom
set_color_pair	scp	sp	Set current color pair
set_foreground	setf	Sf	Set current foreground color
set_left_margin	smgl	ML	Set left margin at current column
set_left_margin_parm	smglp	Zm	Set left (right) margin at column #1 (#2)
set_right_margin	smgr	MR	Set right margin at current column
set_right_margin_parm	smgrp	Zn	Set right margin at column #1
set_tab	hts	st	Set a tab in all rows, current column
set_top_margin	smgt	Zo	Set top margin at current line
set_top_margin_parm	smgtp	Zp	Set top (bottom) margin at line #1 (#2)
set_window	wind	wi	Current window is lines #1 - #2 cols #3 - #4
start_bit_image	sbim	Zq	Start printing bit image graphics
start_char_set_def	scsd	Zr	Start definition of character set
stop_bit_image	rbim	Zs	Stop printing bit image graphics

Table 3-22. String Capabilities (continued)

Variable	TInfo	TCap	Description
`stop_char_set_def`	`rcsd`	`Zt`	Stop definition of character set
`subscript_characters`	`subcs`	`Zu`	List of "subscriptable" characters
`superscript_characters`	`supcs`	`Zv`	List of "superscript-able" characters
`tab`	`ht`	`ta`	Tab to next 8-space hardware tab stop
`these_cause_cr`	`docr`	`Zw`	Printing any of these chars causes cr
`to_status_line`	`tsl`	`ts`	Go to status line
`underline_char`	`uc`	`uc`	Underscore one char and move past it
`up_half_line`	`hu`	`hu`	Half-line up (reverse 1/2 linefeed)
`xoff_character`	`xoffc`	`XF`	XOFF character
`xon_character`	`xonc`	`XN`	XON character
`zero_motion`	`zerom`	`Zx`	No motion for the subsequent character

TERMINFO Compiled File

The `TERM` file is the compiled `terminfo` source file.

Description

The `TERM` file is compiled from `terminfo` source files using the `tic` utility. Compiled files are organized in a directory hierarchy under the first letter of each terminal name. For example, the `vt100` file would have the following pathname.

```
/usr/lib/terminfo/v/vt100
```

The compiled files are read by the `CURSES` routine `setupterm()`.

The following illustration shows the content and order of the compiled file:

```
<magic number><name section size><Boolean section size><number section size>
<string section size><string table size><name section><Boolean section>[0]
<number section><string section><string table>
```

The first six items in the file make up the header.

The header consists of six short integers, stored using VAX/PDP style byte swapping (least-significant byte first). The integers are as follows:

1. magic number (octal 0432)
2. the size, in bytes, of the names section
3. the number of bytes in the Boolean section
4. the number of short integers in the numbers section
5. the number of offsets (short integers) in the strings section
6. the size, in bytes, of the string table

Following the header is the terminal name section that consists of the first line of the `terminfo` definition terminated with an ASCII NUL character.

The terminal name section is followed by the Boolean section, number section, string section, and string table.

The Boolean section consists of a byte for each flag, showing whether the flag is absent, present, or cancelled (a value of 0, 1, or 2 respectively). If necessary, a null byte is inserted between the Boolean and number sections so that the number section begins on an even byte boundary. All short integers are aligned on a short word boundary.

Each capability in the number section is made up of two bytes and stored as a short integer. A value of -1 or -2 indicates a missing or cancelled capability.

Similarly, each capability in the string section is made up of two bytes and stored as a short integer. The value is an offset from the string table. A value of -1 or -2 indicates a missing or cancelled capability. Parameter and padding information is stored in its uninterpreted form. Control or other characters using special notation (^x, \c) are stored in their interpreted form.

The final section of the file is the string table that contains the values of each string in the string section, followed by a null character.

Note	The `setupterm()` routine may expect a different set of capabilities than appears in the file. Unexpected or missing entries may result when the database has been updated since the `CURSES` library was last compiled, or when the program is recompiled more recently than the database.

The first of the following two examples shows a `terminfo` file for a dumb terminal; the second example shows an octal dump of the `TERM` file.

```
000000  032 001 005  \0   %  \0.036  \0   c 001  \r  \0   d   u   m   b
000020   \0  \0 001  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
000040   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0   e  \0  \0
000060   \0  \0  \0  \0  \0  \0   P  \0 377 377 377 377 377 377 377 377
000100  377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
000160  377 377 377 377 005  \0  \a  \0 377 377 377 377 377 377 377 377
000200  377 377 377 377 377 377 377 377  \t  \0 377 377 377 377 377 377
000220  377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
000560  377 377 377 377  \v  \0 377 377 377 377 377 377 377 377 377 377
000600  377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
001460  377 377 377 377 377 377 377 377   d   u   m   b  \0  \a  \0  \r
001500   \0  \n  \0  \n  \0
001505
```

Related Information

`tic` utility, `untic` utility

MPE/iX Reference Supplement (32650-90353)

Implementation Considerations

Identical to UNIX System V

Portability

UNIX System V

4

CURSES

The `CURSES` screen management package consists of routines and macros for creating and modifying input and output to a terminal screen. `CURSES` contains routines for creating windows, highlighting text, writing to the screen, reading from user input, and moving the cursor.

The `CURSES` package is designed to optimize screen update activities. For example, when updating the screen, `CURSES` minimizes the number of characters sent to the terminal to move and update the screen.

`CURSES` is a terminal-independent package, providing a common user interface to a variety of terminal types. Its portability is facilitated by the `TERMINFO` database, which contains a compiled definition of each terminal type. By referring to the database information, `CURSES` gains access to low-level details about individual terminals.

`CURSES` tailors its activities to the terminal type specified by the `TERM` environment variable. The `TERM` environment variable may be set in the MPE/iX shell by entering:

`export TERM=`*terminal_name*

Hewlett-Packard systems default to the `hp2392a` terminal name.

Environment Variables

The following three environment variables are useful, and can be set in the MPE/iX shell:

- TERMINFO
- COLUMNS
- LINES

Refer to the *MPE/iX Shell and Utilities User's Guide* (36431-90002) for more information on the MPE/iX shell.

TERMINFO Environment Variable

If you have an alternate Terminfo database containing terminal types that are not available in the system default database `/usr/lib/terminfo`, you can specify the `TERMINFO` environment variable to point to this alternate database. For example:

```
export TERMINFO=/usr/lib/specialinfo
```

This path specifies the location of the alternate compiled `TERMINFO` database, whose structure consists of directory names 0 to 9 and a to z, each containing compiled terminal definition files for names beginning with the directory letter or number.

The alternate database specified by `TERMINFO` is examined before the system default database. If the terminal type specified by `TERM` cannot be found in either database, the default terminal type `hp2392a` is assumed.

COLUMNS Environment Variable

The `COLUMNS` environment variable is used to set the window width.

For example, to specify a window width smaller than your screen width in situations where your communications line is slow, set the `COLUMNS` environment variable to the number of vertical columns that you want between the left and right margins.

export COLUMNS=*number*

The *number* of columns may be set to a number smaller than the screen size; however, if set larger than the screen or window width, the results are undefined. Currently, the largest screen width possible is 132 columns.

The value set using the `COLUMNS` environment variable takes precedence over the value normally used for the terminal.

LINES Environment Variable

The `LINES` environment variable is used to set the window height.

For example, to specify a window height smaller than your current screen height in situations where your communications line is slow, override the `LINES` environment variable by setting it to a smaller number of horizontal lines.

```
export LINES=number
```

The *number* of lines may be set to a number smaller than the screen height; however, if set larger than the screen or window height, the results are undefined. Currently, the largest screen height possible is 128 lines.

The value set using the `LINES` environment variable takes precedence over the value normally used for the terminal.

Implementation Details

The following routines are not fully implemented:

- `color_pair()`
- `init_color()`
- `init_pair()`
- `napms()`
- `pair_content()`
- `start_color()`

The routines shown in the following table are stubs for the older `TERMCAP` interface and should be replaced by their newer `TERMINFO` counterparts.

TERMCAP	**TERMINFO**
`tgoto()`	`mvcur()`
`tgetent()`	`deleted()`
`tgetflag()`	`tigetflag()`
`tgetnum()`	`tigetnum()`
`tgetstr()`	`tigetstr()`

The following routines have known problems:

`halfdelay()`	improper implementation
`intrflush()`	missing General Terminal Interface support
`nl()`	cannot be disabled with `nonl()`
`nonl()`	cannot be disabled with `nl()`
`nodelay()`	non-blocking input situations
`nocbreak()`	undefined
`scanw()`	undefined
`typeahead()`	non-blocking input situations
`wtime-out(w,0)`	non-blocking input situations

Global Variables

The global variables defined for CURSES are shown in Table 4-1.

Table 4-1. Definitions of Global Variables

Constant	Description
COLORS	Number of colors supported by terminal
COLOR_PAIRS	Number of color pairs supported by terminal
COLS	Number of columns supported by terminal
LINES	Number of lines supported by terminal
boolcodes[ ]	termcap capability names
boolfnames[ ]	Full C names
boolnames[ ]	terminfo capability names
cur_term	Current terminal
curscr	Current screen image
numcodes[ ]	termcap capability codes
numfnames[ ]	Full C names
numfnames[ ]	terminfo capability codes
stdscr	Standard screen supplied by initscr()
strcodes[ ]	termcap capability names
strfnames[ ]	Full C names
strnames[ ]	terminfo capability names
ttytype	Terminal type

Implementation Considerations

The curscr, sdscr, COLS, and LINES constants are identical to XPG/3. The COLORS, COLOR_PAIRS, boolcodes[], boolfnames[], boolnames[], numcodes[], numfnames[], numnames[], strcodes[], strfnames[], and strnames[] constants are UNIX System V implementations.

Portability

The COLORS, COLOR_PAIRS, boolcodes[ ], boolfnames[ ], boolnames[ ], numcodes[ ], numfnames[ ], numnames[ ], strcodes[ ], strfnames[ ], and strnames[ ] constants conform to UNIX System V. The curscr, sdscr, COLS, and LINES constants conform to HP-UX, UNIX System V, and XPG/3. The cur_term and ttytype constants conform to HP-UX and UNIX System V.

Descriptions of CURSES Routines

The following section describes the CURSES routines. The descriptions are presented in sets of related routines. They are arranged alphabetically by the primary routine name; for example, addch is the primary name for the following set of routines:

addch
waddch
mvaddch
mvwaddch

addch
waddch
mvaddch
mvwaddch

The `addch` set of routines is used to add a character (with attributes) to a window.

Syntax

```
int addch(chtype ch);
int waddch(WINDOW *win, chtype ch);
int mvaddch(int y, int x, chtype ch);
int mvwaddch(WINDOW *win, int y, int x, chtype ch);
```

Parameters

ch	The character/attribute pair to be written to the window.
win	A pointer to the window in which the character is to be written.
x	The *x* (column) coordinate of the character's position in the window.
y	The *y* (row) coordinate of the character's position in the window.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred. An attempt was made to write outside the window boundary or after writing a character, the cursor advanced past the scroll region (and `scrollok()` is not set).

Description

A window is made up of foreground and background attributes. All characters except space are part of the foreground. The character and its attributes make up a character/attribute pair defined as a `chtype`. The character is any 16-bit value; the attribute consists of highlighting attributes that affect the appearance of the character on the screen (for example, bold, underline).

Each time that a character, other than a space, is written to a window with `waddch()`, `wprintw()`, or `waddstr()`, a bitwise `OR` operation is performed between the `chtype` (foreground character with its attributes), the current foreground attributes of the window, and the current background attributes of the window. The current foreground attributes are set with the `wattrset()`, `wattron()`, and `wattroff()` routines; the current background attributes are set with the `wbgdset()` routine.

When spaces are written to the screen, the background character and attributes replace the space. For example, if the background attribute and character is

```
A_UNDERLINE|'*'
```

text written to the window appears underlined, and the spaces appear as underlined asterisks.

After the `OR` operation, each character written retains the new foreground and background attributes that it has obtained. This allows the character to be copied as is to a window with the `waddchstr()` or `insch()` routines.

The `addch()` routine writes a character to the `stdscr` window at the current cursor position and advances the cursor. The `waddch()` routine performs an identical action, but writes the character to the window specified by *win*. The `mvaddch()` and `mvwaddch()` routines write the character to the position indicated by the x (column) and y (row) parameters. The `mvaddch()` routine writes the character to the `stdscr` window, while `mvwaddch()` writes the character to the window specified by *win*.

If the character is a newline, carriage return, backspace, or tab, the cursor is moved appropriately. The cursor is moved to the next tab stop for each tab character (tab stops are eight characters apart). If the character is a control character other than those previously mentioned, the character is written using $\hat{}x$ notation, where x is a printable character. If the character is written to the last character position on a line, a newline is generated automatically. If the character is written to the last character position of a scrolling region and `scrollok()` is enabled, the scrolling region is scrolled up one line (see `wsetscrreg()`).

Individual characters can be highlighted by performing a bitwise `OR` operation between the character and one or more of the constants shown in Table 4-2.

Table 4-2. Constant Values for Highlighting Attributes

Constant	Description
`A_ALTCHARSET`	Alternate character set
`A_ATTRIBUTES`	Attribute mask
`A_BLINK`	Blinking
`A_BOLD`	Bold
`A_CHARTEXT`	Character mask
`A_COLOR`	Color mask
`A_DIM`	Dim
`A_INVIS`	Invisible
`A_NORMAL`	Disable attributes
`A_PROTECT`	No display
`A_REVERSE`	Reverse video
`A_STANDOUT`	Highlights specific to terminal
`A_UNDERLINE`	Underline
`COLOR_PAIR`(n)	Color pair number n
`PAIR_NUMBER`(a)	Pair number for `COLOR_PAIR`(n)

The characters shown in Table 4-3 are defined as constants in `CURSES`.

Table 4-3. Constant Values for Characters

<table>
<tr><th>Constant</th><th>Character</th><th>Description</th></tr>
<tr><td>ACS_VLINE</td><td>|</td><td>Vertical line</td></tr>
<tr><td>ACS_HLINE</td><td>-</td><td>Horizontal line</td></tr>
<tr><td>ACS_ULCORNER</td><td>+</td><td>Upper-left corner</td></tr>
<tr><td>ACS_URCORNER</td><td>+</td><td>Upper-right corner</td></tr>
<tr><td>ACS_BLCORNER</td><td>+</td><td>Bottom-left corner</td></tr>
<tr><td>ACS_BRCORNER</td><td>+</td><td>Bottom-right corner</td></tr>
<tr><td>ACS_RTEE</td><td>+</td><td>Right tee (-|)</td></tr>
<tr><td>ACS_LTEE</td><td>+</td><td>Left tee (|-)</td></tr>
<tr><td>ACS_BTEE</td><td>+</td><td>Bottom tee ()</td></tr>
<tr><td>ACS_TTEE</td><td>+</td><td>Top tee (T)</td></tr>
<tr><td>ACS_CHECK</td><td>!</td><td>Check mark</td></tr>
<tr><td>ACS_PLUS</td><td>+</td><td>Plus</td></tr>
<tr><td>ACS_DIAMOND</td><td>+</td><td>Diamond</td></tr>
<tr><td>ACS_CKBOARD</td><td>:</td><td>Checker board</td></tr>
<tr><td>ACS_DEGREE</td><td>’</td><td>Degree sign</td></tr>
<tr><td>ACS_PLMINUS</td><td>#</td><td>Plus/Minus</td></tr>
<tr><td>ACS_BULLET</td><td>o</td><td>Bullet</td></tr>
<tr><td>ACS_LARROW</td><td><</td><td>Left arrow</td></tr>
<tr><td>ACS_RARROW</td><td>></td><td>Right arrow</td></tr>
<tr><td>ACS_DARROW</td><td>v</td><td>Down arrow</td></tr>
<tr><td>ACS_UARROW</td><td>ˆ</td><td>Up arrow</td></tr>
<tr><td>ACS_BOARD</td><td>#</td><td>Board of squares</td></tr>
<tr><td>ACS_LANTERN</td><td>#</td><td>Lantern symbol</td></tr>
<tr><td>ACS_BLOCK</td><td>#</td><td>Solid square block</td></tr>
</table>

Note The `addch()`, `mvaddch()`, and `mvwaddch()` routines are macros.

Implementation Considerations

Identical to XPG/3.

See Also

`winsch()`, `nl()`, `nonl()`, `scrollok()`, `wattron()`, `wattroff()`, `wattrset()`, `wbkgdset()`, `wprintw()`, `wscrl()`, `wsetscrreg()`

Portability

HP-UX, UNIX System V, XPG/3

addchstr
waddchstr
addchnstr
waddchnstr
mvaddchstr
mvwaddchstr
mvaddchnstr
mvwaddchnstr

The `addchstr` set of routines is used to copy a character string (with attributes) to a window.

Syntax

```
#include <curses.h>

int addchstr(chtype *chstr);
int waddchstr(WINDOW *win, chtype *chstr);

int addchnstr(chtype *chstr, int n);
int waddchnstr(WINDOW *win, chtype *chstr, int n);

int mvaddchstr(int y, int x, chtype *chstr);
int mvwaddchstr(WINDOW *win, int y, int x, chtype *chstr);

int mvaddchnstr(int y, int x, chtype *chstr, int n);
int mvwaddchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);
```

Parameters

chstr — A pointer to the `chtype` string to be copied to the window.

n — The maximum number of characters to be copied from *chstr*. If *n* is less than 0, the entire string is written, or as much of it as fits on the line.

win — A pointer to the window to which the string is to be copied.

x The *x* (column) coordinate of the starting position of *chstr* in the window.

y The *y* (row) coordinate of the starting position of *chstr* in the window.

Return Values

`OK` Successful completion.

`ERR` An error occurred.

Description

The `addchstr()` routine copies the `chtype` character string to the `stdscr` window at the current cursor position. The `waddchstr()` routine performs the identical action, but writes to the window specified by *win*. The `mvaddchstr()` and `mvwaddchstr()` routines copy the character string to the starting position indicated by the *x* (column) and *y* (row) parameters (the former to the `stdscr` window; the latter to window *win*).

The `addchnstr()`, `waddchnstr()`, `mvaddchnstr()`, and `mvwaddchnstr()` routines write *n* characters to the window, or as many as will fit on the line. If *n* is less than 0, the entire string is written, or as much of it as fits on the line. The former two routines place the string at the current cursor position; the latter two commands use the position specified by the *x* and *y* parameters.

These routines differ from the `waddnstr()` set of routines in several important respects. First, the position of the cursor is not advanced after the string is written to the window. Second, these routines are faster because they copy the string into the window without performing checks such as line wrapping on a newline; instead, the string is truncated if it does not fit on the line. Third, the current foreground and background window attributes are not combined with the character; only the attributes that are already part of the `chtype` character are used.

Note All routines except `waddchnstr()` are macros.

Implementation Considerations

UNIX System V implementation

See Also

`waddch()`, `waddnstr()`, `wattrset()`

Portability

UNIX System V

addstr
waddstr
addnstr
waddnstr
mvaddstr
mvwaddstr
mvaddnstr
mvwaddnstr

The `addstr` set of routines is used to add a character string (with attributes) to a window.

Syntax

```
#include <curses.h>

int addstr(char *str);
int waddstr(WINDOW *win, char *str);

int addnstr(char *str, int n);
int waddnstr(WINDOW *win, char *str, int n);

int mvaddstr(int y, int x, char *str);
int mvwaddstr(WINDOW *win, int y, int x, char *str);

int mvaddnstr(int y, int x, char *str, int n);
int mvwaddnstr(WINDOW *win, int y, int x, char *str, int n);
```

Parameters

str A pointer to the character string that is to be written to the window.

win A pointer to the window in which the string is to be written.

x The x (column) coordinate of the starting position of *str* in the window.

y The *y* (row) coordinate of the starting position of *str* in the window.

Return Values

`OK` Successful completion.

`ERR` An error occurred. An attempt was made to write outside the window boundary.

Description

The `addstr()` routine writes a null-terminated character string to the `stdscr` window at the current cursor position and advances the cursor. The `waddstr()` routine performs an identical action, but writes the character to the window specified by *win*. The `mvaddstr()` and `mvwaddstr()` routines write the character string to the position indicated by the *x* (column) and *y* (row) parameters (the former to the `stdscr` window; the latter to window *win*).

The functionality of these routines is equivalent to calling the corresponding `waddch()` set of routines once for each character in the string. Refer to `waddch()` for a complete description of the interaction between the foreground and background attributes of the window and the character written. Note that these routines differ from the `waddchnstr()` set of routines in that the latter copy the string as is (without combining each character with the foreground and background attributes of the window).

The `addnstr()`, `waddnstr()`, `mvaddnstr()`, and `mvwaddnstr()` routines write at most *n* characters to the window. If *n* is less than 0, the entire string is written. The former two routines place the characters at the current cursor position; the latter two commands use the position specified by the *x* and *y* parameters.

Note All routines except `waddnstr()` are macros.

Implementation Considerations

The `addstr()`, `waddstr()`, `mvaddstr()`, and `mvwaddstr()` are identical to XPG/3. The `addnstr()`, `waddnstr()`, `mvaddnstr()`, and `mvwaddnstr()` routines are UNIX System V implementations.

See Also

`waddch()`, `waddchnstr()`

Portability

The `addstr()`, `waddstr()`, `mvaddstr()`, and `mvwaddstr()` routines conform to HP-UX, UNIX System V, and XPG/3. The `addnstr()`, `waddnstr()`, `mvaddnstr()`, and `mvwaddnstr()` routines conform to UNIX System V.

attroff
wattroff
attron
wattron
attrset
wattrset
standend
wstandend
standout
wstandout

The `attroff` set of routines is used to change the foreground window attributes.

Syntax

```
#include <curses.h>

int attroff(chtype attrs);
int wattroff(WINDOW *win, chtype attrs);

int attron(chtype attrs);
int wattron(WINDOW *win, chtype attrs);

int attrset(chtype attrs);
int wattrset(WINDOW *win, chtype attrs);

int standend();
int wstandend(WINDOW *win);

int standout();
int wstandout(WINDOW *win);
```

Parameters

attrs	The foreground window attributes to be added or removed.
win	A pointer to the window in which attribute changes are to be made.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `attroff()` and `attron()` routines remove and add, respectively, the specified foreground window attributes of `stdscr`. These routines only affect the attributes specified; attributes that existed before the call retain their values. The `wattroff()` and `wattron()` routines remove or add the specified attributes for window *win*.

The `attrset()` and `wattrset()` routines change the specified foreground window attributes of `stdscr` and *win* to new values; the old values are not retained.

The attributes shown in Table 4-4 are defined in `curses.h` and can be combined with the bitwise `OR` operator.

Table 4-4. Constant Values for Highlighting Attributes

Constant	Description
`A_ALTCHARSET`	Alternate character set
`A_ATTRIBUTES`	Attribute mask
`A_BLINK`	Blinking
`A_BOLD`	Bold
`A_CHARTEXT`	Character mask
`A_COLOR`	Color mask
`A_DIM`	Dim
`A_INVIS`	Invisible
`A_NORMAL`	Disable attributes
`A_PROTECT`	No display
`A_REVERSE`	Reverse video
`A_STANDOUT`	Highlights specific to terminal
`A_UNDERLINE`	Underline
`COLOR_PAIR(`*n*`)`	Color-pair number *n*
`PAIR_NUMBER(`*a*`)`	Pair number for `COLOR_PAIR(`*n*`)`

The `standend()` routine is equivalent to `attrset(A_NORMAL)`. Similarly, the `wstandend()` routine is equivalent to `wattrset(win, A_NORMAL)`.

The `standout()` and `wstandout()` routines are equivalent to `attron(A_STANDOUT)` and `wattron(win, A_STANDOUT)`, respectively.

`CURSES` applies the current foreground attributes when writing characters to a window with the `waddch()`, `waddstr()`, or `wprintw()` routines.

The following example prints some text using the current foreground attributes, adds underlining, changes the attributes, prints more text, then changes the attributes back.

```
printw("This word is");
attrset(A_UNDERLINE);
printw("underlined.");
attrset(A_NORMAL);
printw("This is back to normal text.\n");
refresh();
```

Note All of these routines are macros.

Implementation Considerations

Identical to XPG/3 except for color support

See Also

`init_color()`, `init_pair()`, `start_color()`, `wbkgd()`, `wbkgdset()`

Portability

HP-UX, UNIX System V, XPG/3

baudrate

The `baudrate` routine returns the terminal baud rate.

Syntax

```
#include <curses.h>

int baudrate();
```

Return Values

The terminal's baud rate is returned in bits per second.

Description

The `baudrate()` routine returns the terminal's data communication line and output speed in bits per second (for example, 9600).

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

beep
flash

The `beep` and `flash` routines activate the audio-visual alarm.

Syntax

```
#include <curses.h>

int beep();
int flash();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred. The terminal does not support either capability.

Description

The `beep()` and `flash()` routines produce an audio and visual alarm on the terminal, respectively. If the terminal has the capability, `beep()` sounds a bell or beep, and `flash()` flashes the screen. One alarm is substituted for another if the terminal does not support the capability called. For example, a call to `beep()` for a terminal without that capability results in a flash and vice versa.

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

bkgdset
wbkgdset
bkgd
wbkgd

The `bkgdset` set of routines is used to set the background character (and attributes) of a window.

Syntax

```
#include <curses.h>

void bkgdset(chtype ch);
void wbkgdset(WINDOW *win, chtype ch);

int bkgd(chtype ch);
int wbkgd(WINDOW *win, chtype ch);
```

Parameters

ch	A pointer to the background character to be set.
win	A pointer to the window in which the background character is to be set.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

A window is made up of foreground and background attributes. All characters except space are part of the foreground. The character and its attributes make up a character/attribute pair defined as a `chtype`. The character is any 16-bit value; the attribute consists of highlighting attributes that affect the appearance of the character on the screen (for example, bold, underline).

Each time a character, other than a space, is written to a window with `waddch()`, `wprintw()`, or `waddstr()`, a bitwise `OR` operation is performed between the `chtype` (foreground character with its attributes), the current foreground attributes of the window, and the current background attributes of the window. The current foreground attributes are set with `wattrset()`, `wattron()`, and `wattroff()`; the current background attributes are set with `wbgdset()`. When spaces are written to the screen, the background character and attributes replace the space. For example, if the background attribute and character is

```
A_UNDERLINE|'*'
```

text written to the window appears underlined and the spaces appear as underlined asterisks.

After the `OR` operation, each character written retains the new foreground and background attributes that it has obtained. This allows the character to be copied "as is" to a window with the `waddchstr()` or `insch()` routines.

The `bkgdset()` routine sets the current background character and attributes for the `stdscr` window; `wbgdset()` sets the current background character and attributes for window *win*. You must specify the complete character/attribute pair; for example:

```
bkgdset(A_BOLD|' ');
```

sets the background attribute as bold and the background character as a space. The default background character/attribute pair is

```
bkgdset(A_NORMAL|' ');
```

Note The current background character and attributes are written to the window by the `wclear()`, `werase()`, `cltroeol()`, and `cltrobot()` routines as well as any other routines that insert blanks. If a background character is not supplied (that is, only an attribute is given), results are undefined.

The `bkgd()` and `wbkgd()` routines update the entire window (`stdscr` and *win*, respectively) with the supplied background and perform a `wbkgdset()`.

Note The `bkgd()`, `wbkgd()`, and `bkgdset()` routines are macros.

Implementation Considerations

UNIX System V implementation

See Also

`addch()`, `attroff()`, `attron()`, `waddchstr()`, `wattrset()`, `winsch()`

Portability

UNIX System V

border
box
wborder

The `border` set of routines is used to add a border to a window.

Syntax

```
#include <curses.h>

int border(chtype ls, chtype rs, chtype ts, chtype bs,
   chtype tl, chtype tr, chtype bl, chtype br);
int wborder(WINDOW *win, chtype ls, chtype rs,
   chtype ts, chtype bs, chtype tl, chtype tr,
   chtype bl, chtype br);

int box(WINDOW *win, chtype verch, chtype horch);
```

Parameters

bl The character and attributes used for the bottom-left corner of the border.

br The character and attributes used for the bottom-right corner of the border.

bs The character and attributes used for the bottom of the border.

horch The character and attributes used for the top and bottom rows of the box.

ls The character and attributes used for the left side of the border.

rs The character and attributes used for the right side of the border.

tl The character and attributes used for the top- left corner of the border.

tr The character and attributes used for the top- right corner of the border.

ts The character and attributes used for the top of the border.

verch The character and attributes used for the left and right columns of the box.

win The pointer to the window in which the border or box is to be drawn.

Return Values

`OK` Successful completion.

`ERR` An error occurred.

Description

The `border()`, `wborder()`, and `box()` routines draw a border around the specified window. A parameter with the value of zero is replaced by the default value as defined in `curses.h`. The constant values for a border are shown in Table 4-5.

Table 4-5. Constant Values for Borders

Parameter	Constant Used	Value
verch	ACS_VLINE	|
horch	ACS_HLINE	-
ls	ACS_VLINE	|
rs	ACS_VLINE	|
ts	ACS_HLINE	-
bs	ACS_HLINE	-
bl	ACS_BLCORNER	+
br	ACS_BRCORNER	+
tl	ACS_ULCORNER	+
tr	ACS_URCORNER	+

The call

`box` (*win*, *verch*, *horch*)

is a short form for

`wborder(`*win*, *verch*, *verch*, *horch*, *horch*, `0, 0, 0, 0)`

When the window is boxed, the bottom and top rows and right and left columns are unavailable for text.

Note The `border()` and `box()` routines are macros.

Implementation Considerations

The `box()` routine is identical to XPG/3. The `border()` and `wborder()` routines are UNIX System V implementations.

See Also

`waddch()`, `wattrset()`

Portability

The `box()` routine conforms to HP-UX, UNIX System V, and XPG/3. The `border()` and `wborder()` routines conform to UNIX System V.

cbreak
nocbreak

The `cbreak` and `nocbreak` routines enable and disable the character-mode operation.

Syntax

```
#include <curses.h>

int cbreak();
int nocbreak();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `cbreak()` and `nocbreak()` routines enable and disable character-mode operation, respectively. When enabled, characters typed by the user are immediately processed by the program. When disabled, the terminal driver is placed into line canonical input mode, which buffers typed characters (until a newline or carriage return are typed) and handles `erase()` and `kill()` character processing. These routines do not affect flow control or interrupt characters.

The terminal may or may not be in character mode operation initially. Most interactive programs require `cbreak()` to be enabled.

Implementation Considerations

Identical to XPG/3

See Also

`wgetch()`, `halfdelay()`, `nodelay()`, `raw()`, `wtimeout()`

Portability

HP-UX, UNIX System V, XPG/3

clear
wclear

The `clear` and `wclear` routines are used to clear the window.

Syntax

```
#include <curses.h>

int clear();
int wclear(WINDOW *win);
```

Parameters

win	A pointer to the window that is to be cleared.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `clear()` routine clears `stdscr`, destroying its previous contents. The `wclear()` routine performs the same action, but clears the window specified by *win* instead of `stdscr`. These routines are similar to `erase()` and `werase()` except they also call `clearok()`. The `clearok()` routine clears and redraws the entire screen on the next call to `wrefresh()` for the window.

The current background character (and attributes) is used to clear the screen.

Note The `clear()` routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`clearok()`, `wbkgdset()`, `wclrtobot()`, `wclrtoeol()`, `werase()`

Portability

HP-UX, UNIX System V, XPG/3

clearok

The `clearok` routine is used to clear and redraw the window with the next refresh.

Syntax

```
#include <curses.h>

int clearok(WINDOW *win, bool bf);
```

Parameters

win	A pointer to the window that is to be cleared and refreshed.
bf	A Boolean expression.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

If *bf* is `TRUE`, `clearok()` clears and redraws the entire screen on the next call to `wrefresh()`. If *win* is `curscr`, the next call to `wrefresh()` for any window clears and redraws the screen.

Implementation Considerations

Identical to XPG/3

See Also

`wbkgdset()`, `wclear()`, `werase()`, `wrefresh()`

Portability

HP-UX, UNIX System V, XPG/3

clrtobot
wclrtobot

The `clrtobot` and `wclrtobot` routines are used to clear to the end of the window.

Syntax

```
#include <curses.h>

int clrtobot();
int wclrtobot(WINDOW *win);
```

Parameters

win	A pointer to the window that is to be cleared.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `clrtobot()` routine clears all characters in the `stdscr` window from the cursor to the end of the window. The `wclrtobot()` routine performs the same action in the window specified by *win* instead of in `stdscr`. The current background character (and attributes) is used to clear the screen.

Note The `clrtobot()` routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`clearok()`, `wbkgdset()`, `wclear()`, `wcrltoeol()`, `werase()`

Portability

HP-UX, UNIX System V, XPG/3

clrtoeol
wclrtoeol

The `clrtoeol` and `wclrtoeol` routines are used to clear to end of line.

Syntax

```
#include <curses.h>

int clrtoeol();
int wclrtoeol(WINDOW *win);
```

Parameters

win	A pointer to the window in which to clear to the end of the line.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `clrtoeol()` routine clears the current line from the cursor to the right margin in the `stdscr` window. The `wclrtoeol()` routine performs the same action, but in the window specified by *win* instead of `stdscr`. The current background character (and attributes) is used to clear the screen.

Note The `clrtoeol()` routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`clearok()`, `wbkgdset()`, `wclear()`, `wclrtobot()`, `werase()`

Portability

HP-UX, UNIX System V, XPG/3

color_pair

Note The `color_pair` routine is not implemented at this time.

FINAL TRIM SIZE : 7.0 in x 8.5 in

copywin

The `copywin` routine is used to overlay or overwrite any portion of window.

Syntax

```
#include <curses.h>

int copywin(WINDOW *srcwin, WINDOW *dstwin, int sminrow, int smincol,
int dminrow, int dmincol, int dmaxrow, int dmaxcol, int overlay);
```

Parameters

srcwin	A pointer to the source window to be copied.
dstwin	A pointer to the destination window to be overlayed or overwritten.
smincol	The column coordinate of the upper-left corner of the rectangular area on the source window to be copied.
sminrow	The row coordinate of the upper-left corner of the rectangular area on the source window to be copied.
dmincol	The column coordinate of the upper-left corner of the rectangular area on destination window to be overlayed or overwritten.
dminrow	The row coordinate of the upper-left corner of the rectangular area on the destination window to be overlayed or overwritten.
dmaxcol	The column coordinate of the lower-right corner of the rectangular area on the destination window to be overlayed or overwritten.
dmaxrow	The row coordinate of the lower-right corner of the rectangular area on the destination window to be overlayed or overwritten.
overlay	A true or false value that determines whether the destination window is overlayed or overwritten.

Return Values

`OK` Successful completion.

`ERR` An error occurred.

Description

The `copywin()` routine overlays or overwrites windows similiar to the `overlay()` and `overwrite()` functions; however, `copywin()` allows a finer degree of control on what portion of the window to overlay or overwrite.

The parameters *smincol* and *sminrow* specify the upper-left corner of the rectangular area of the source window to be copied. The *dminrow* and *dmincol* parameters specify the upper-left corner of the rectangular area of the destination window to which the specified portion of the source is to be copied. The *dmaxrow* and *dmaxcol* parameters specify the bottom-right corner of the rectangular area of the destination window to which the specified portion of the source is to be copied.

If *overlay* is `TRUE`, only nonblank characters are copied to the destination window; if `FALSE`, all characters are copied.

Implementation Considerations

UNIX System V implementation

See Also

`overlay()`, `overwrite()`

Portability

UNIX System V

curs_set

The `curs_set` routine is used to set the visibility of the cursor.

Syntax

```
#include <curses.h>

int curs_set(int visibility);
```

Parameters

visibility	A value of 0 (invisible), 1 (normal), or 2 (very visible).

Return Values

On success, previous cursor visibility is returned; `ERR` is returned if the requested visibility is not supported.

Description

The `curs_set()` routine sets the visibility of the cursor to invisible (0), normal (1), or very visible (2).

Implementation Considerations

UNIX System V implementation

Portability

UNIX System V

def_prog_mode
def_shell_mode

The def_prog_mode and def_shell_mode routines are used to save terminal modes.

Syntax

```
#include <curses.h>

int def_prog_mode();
int def_shell_mode();
```

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The def_prog_mode() and def_shell_mode() routines save the current terminal modes as "program" (within CURSES) or "shell" (outside CURSES). These are used by the reset_prog_mode() and reset_shell_mode() routines. The modes are saved automatically by the initscr(), newterm(), and setupterm() routines.

These routines can also be used outside CURSES with terminfo routines.

Implementation Considerations

Identical to XPG/3

See Also

initscr(), newterm(), setupterm(), reset_prog_mode(), reset_shell_mode()

Portability

UNIX System V, XPG/3

del_curterm

The `del_curterm` routine is used to free space pointed to by `TERMINAL` (interface to `TERMINFO`).

Syntax

```
#include <curses.h>

int del_curterm(TERMINAL *oterm);
```

Parameters

oterm	The terminal type for which to free space.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `del_curterm()` routine is a low-level routine only used outside of `CURSES` when the program has to deal directly with the `TERMINFO` database to handle certain terminal capabilities. The use of appropriate `CURSES` routines is recommended in all other situations.

The `del_curterm()` routine frees the space pointed to by *oterm*. If *oterm* and the *cur_term* variable are the same, the `TERMINFO` *Boolean*, *numeric*, or *string* variables refer to invalid memory locations until you call `setupterm()` and specify a new terminal type.

Implementation Considerations

UNIX System V implementation

See Also

`set_curterm()`

Portability

UNIX System V

delay_output

The `delay_output` routine is used to delay output.

Syntax

```
#include <curses.h>

int delay_output(int ms);
```

Parameters

ms	The number of milliseconds to delay the output.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `delay_output()` routine delays output for *ms* milliseconds by inserting pad characters in the output stream.

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

delch
wdelch
mvdelch
mvwdelch

The `delch` set of routines is used to remove a character.

Syntax

```
#include <curses.h>

int delch();
int wdelch(WINDOW *win);

int mvdelch(int y, int x);
int mvwdelch(WINDOW *win, int y, int x);
```

Parameters

x	The *x* (column) coordinate of the position of the character to be removed.
y	The *y* (row) coordinate of the position of the character to be removed.
win	A pointer to the window containing the character to be removed.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `delch()` and `wdelch()` routines delete the character at the current cursor position from `stdscr` and *win*, respectively. All remaining characters on the same line to the right of the deleted character are moved left one character. The last character on the line becomes a space; characters on other lines are not affected.

The `mvdelch()` and `mvwdelch()` routines delete the character at the position specified by the x and y parameters; the former deletes the character from `stdscr`; the latter from *win*.

Note The `delch()`, `mvdelch()`, and `mvwdelch()` routines are macros.

Implementation Considerations

Identical to XPG/3

See Also

`wbgdset()`

Portability

HP-UX, UNIX System V, XPG/3

deleteln
wdeleteln

The `deleteln` and `wdeleteln` routines are used to remove a line.

Syntax

```
#include <curses.h>

int deleteln();
int wdeleteln (WINDOW *win);
```

Parameters

win	A pointer to the window from which the line is removed.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `deleteln()` and `wdeleteln()` routines delete the line containing the cursor from `stdscr` and *win*, respectively. All lines below the one deleted are moved up one line. The last line of the window becomes blank. The position of the cursor is unchanged.

Note These routines are macros.

Implementation Considerations

Identical to XPG/3

See Also

`winsdeln()`, `winsertln()`, `wbkgdset()`

Portability

HP-UX, UNIX System V, XPG/3

delscreen

The `delscreen` routine is used to free space associated with the `SCREEN` data structure.

Syntax

```
#include <curses.h>

int delscreen(SCREEN *sp);
```

Parameters

sp	A pointer to the screen structure for which to free space.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `delscreen()` routine frees space associated with the `SCREEN` data structure. This routine should be called after `endwin()` if a `SCREEN` data structure is no longer needed.

Implementation Considerations

UNIX System V implementation

See Also

`endwin()`, `initscr()`, `newterm()`

Portability

UNIX System V

delwin

The `delwin` routine is used to delete a window.

Syntax

```
#include <curses.h>

int delwin(WINDOW *win);
```

Parameters

win	A pointer to the window that is to be deleted.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `delwin()` routine deletes the specified window, freeing up the memory associated with it.

Note If you delete a parent window without deleting its subwindows and then try to manipulate the subwindows, you may encounter odd results.

Implementation Considerations

Identical to XPG/3

See Also

`newwin()`, `subwin()`, `derwin()`

Portability

HP-UX, UNIX System V, XPG/3

derwin

The `derwin` routine is used to create a subwindow relative to parent window.

Syntax

```
#include <curses.h>

WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int begin_y,
   int begin_x);
```

Parameters

orio	A pointer to the parent window for the newly created subwindow.
nlines	The number of lines in the subwindow.
ncols	The number of columns in the subwindow.
begin_y	The y (row) coordinate of the upper-left corner of the subwindow, relative to the parent window.
begin_x	The x (column) coordinate of the upper-left corner of the subwindow, relative to the parent window.

Return Values

On success, a pointer to the new window structure is returned; otherwise, a null pointer is returned.

Description

The `derwin()` routine creates a subwindow within window *orig*, with the specified number of lines and columns, and upper left corner positioned at *begin_x*, *begin_y* relative to window *orig*. A pointer to the new window structure is returned.

The original window and subwindow share character storage of the overlapping area. (Each window maintains its own pointers, cursor location, and other items.) This means that characters and attributes are identical in overlapping areas regardless of which window characters are written to.

When using subwindows, it is often necessary to call `touchwin()` before `wrefresh()` to maintain proper screen contents.

Note The `subwin()` routine creates a subwindow in exactly the same way, but allows you to specify coordinates relative to the physical screen.

Implementation Considerations

UNIX System V implementation

See Also

`newwin()`, `subwin()`, `touchwin()`, `delwin()`

Portability

UNIX System V

dupwin

The `dupwin` routine is used to create a duplicate of a window.

Syntax

```
#include <curses.h>

WINDOW *dupwin(WINDOW, *win);
```

Parameters

win A pointer to the window that is to be duplicated.

Return Values

On success, a pointer to new window structure is returned; otherwise, a null pointer is returned.

Description

The `dupwin()` routine creates a duplicate of the window *win*. A pointer to the new window structure is returned.

Implementation Considerations

UNIX System V implementation

See Also

`derwin()`, `newwin()`, `subwin()`

Portability

UNIX System V

echo
noecho

The `echo` and `noecho` routines are used to enable and disable terminal echo.

Syntax

```
#include <curses.h>

int echo();
int noecho();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `echo()` and `noecho()` routines enable and disable the terminal echo, respectively. When enabled, characters received by `getch()` are echoed back to the terminal. When disabled, characters are transferred to the program without echoing them to the terminal display. The program may instead echo the characters to an area of the screen controlled by the program or may not echo the characters at all. Terminal echo is enabled, by default.

Subsequent calls to `echo()` or `noecho()` do not flush type-ahead.

Note The tty driver `echo` is disabled by `initscr()` and `newterm()`. All echoing is controlled by `CURSES`.

Implementation Considerations

Identical to XPG/3

See Also

`wgetch()`, `wgetstr()`, `wscanw()`

Portability

HP-UX, UNIX System V, XPG/3

echochar
wechochar

The `echochar` and `wechochar` routines are used to add a character and refresh the window.

Syntax

```
#include <curses.h>

int echochar(chtype ch);
int wechochar(WINDOW *win, chtype ch);
```

Parameters

win	A pointer to the window in which the character is to be added.
ch	A pointer to the character to be written to the window.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `echochar()` and `wechochar()` routines produce the same effect as a call to `addch()` followed by a call to `refresh()`, or a call to `waddch()` followed by a call to `wrefresh()`, respectively.

Implementation Considerations

UNIX System V implementation

See Also

`waddch()`, `wrefresh()`

Portability

UNIX System V

endwin
isendwin

The endwin and isendwin routines are used to restore the initial terminal environment.

Syntax

```
#include <curses.h>

int endwin();
int isendwin();
```

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The endwin() routine restores tty modes, resets the terminal, and moves the cursor to the lower-left corner of the screen. This routine should be called before exiting or escaping CURSES temporarily. To resume CURSES after a temporary escape, call the wrefresh() or doupdate() routines.

If the program interacts with multiple terminals, call endwin() for each terminal.

The isendwin() routine returns TRUE if endwin() has been called without subsequent calls to wrefresh() and returns FALSE otherwise.

Implementation Considerations

The endwin() routine is identical to XPG/3. The isendwin() routine is a UNIX System V implementation.

See Also

`doupdate()`, `wrefresh()`

Portability

The `endwin()` routine conforms to HP-UX, UNIX System V, and XPG/3. The `isendwin()` routine conforms to UNIX System V.

erase
werase

The erase and werase routines are used to erase a window.

Syntax

```
#include <curses.h>

int erase();
int werase(WINDOW *win);
```

Parameters

win	A pointer to the window that you want to erase.

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The erase() routine erases the contents of the stdscr window, destroying its previous contents. The werase() routine performs the same action, but erases the content of *win* instead of stdscr. The current background character (and attributes) is used to erase the screen.

Note The erase() routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`clearok()`, `wbkgdset()`, `wclear()`, `wclrtobot()`, `wclrtoeol()`

Portability

HP-UX, UNIX System V, XPG/3

erasechar

The `erasechar` routine is used to return the current `ERASE` character.

Syntax

```
#include <curses.h>

char erasechar();
```

Return Values

The terminal's current `ERASE` character is returned.

Description

The `erasechar()` routine returns the user's choice of `ERASE` character from the tty driver. This character is used to delete the previous character during keyboard input. The returned value can be used when including deletion capability in interactive programs.

Implementation Considerations

Identical to XPG/3

See Also

`wgetnstr()`

Portability

HP-UX, UNIX System V, XPG/3

flushinp

The `flushinp` routine is used to discard type-ahead characters.

Syntax

```
#include <curses.h>

int flushinp();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `flushinp()` routine discards all type-ahead characters (characters typed by the user, but not yet processed by `CURSES`).

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

getch
wgetch
mvgetch
mvwgetch
ungetch

The `getch`, `wgetch`, `mvgetch`, `mvwgetch`, and `ungetch` routines are used to get a character from the keyboard.

Syntax

```
#include <curses.h>

int getch();
int wgetch (WINDOW *win);

int mvgetch(int y, int x);
int mvwgetch(WINDOW *win, int y, int x);

int ungetch(int ch);
```

Parameters

ch — The character to be put back in the input queue for the next call to `getch()`.

x — The *x* (column) coordinate for the position of the character to be read.

y — The *y* (row) coordinate for the position of the character to be read.

win — A pointer to the window associated with the terminal from which the character is to be read.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred. The `nodelay()` or `wtimeout(0)` routine is set, and no input is ready.

Description

The `getch()` and `wgetch()` routines get a character from the terminal associated with the window `stdscr`or window *win*, respectively. The `mvgetch()` and `mvwgetch()` routines move the cursor to the position specified in `stdscr` or *win*, respectively, then get a character.

If the window is not a pad and has been changed since the last call to `wrefresh()`, `getch()` calls `wrefresh()` to update the window before the next character is read.

The setting of certain routines affects how `getch()` works. For example, if `cbreak()` is set, characters typed by the user are immediately processed. If `halfdelay()` is set, `getch()` waits until a character is typed or returns `ERR` if no character is typed within the specified timeout period. This timeout can also be specified for individual windows with the `delay` parameter of `wtimeout()`. A negative value waits for input; a value of 0 returns `ERR` if no input is ready; a positive value blocks until input arrives or the time specified expires (in which case `ERR` returns). If `nodelay()` is set, `ERR` is returned if no input is waiting; if not set, `getch()` waits until input arrives. Each character will be echoed to the window unless `noecho()` has been set.

If keypad handling is enabled (`keypad()` is `TRUE`), the token for the function key is returned. If a character is received that could be the beginning of a function key (for example, ESC), an interbyte timer is set. If the remainder of the sequence is not received before the time expires, the character is passed through; otherwise, the value of the function key is returned. If `notimeout()` is set, the interbyte timer is not set.

Note The ESCAPE key is typically a prefix key used with function keys. Since prefix keys used with function keys should not be used as a single character, ensure that you do not use the ESCAPE key as a single character.

Table 4-6 shows a list of tokens for the function keys that are returned by `getch()` if keypad handling is enabled. (Some terminals may not support all tokens.)

Table 4-6. Constant Values for Function Keys

Constant	Description
`KEY_BREAK`	Break key
`KEY_DOWN`	The down arrow key
`KEY_UP`	The up arrow key
`KEY_LEFT`	The left arrow key
`KEY_RIGHT`	The right arrow key
`KEY_HOME`	Home key
`KEY_BACKSPACE`	Backspace
`KEY_F0`	Function keys. Space for 64
`KEY_F(`*n*`)`	(KEY_F0+(*n*)) keys is reserved
`KEY_DL`	Delete line
`KEY_IL`	Insert line
`KEY_DC`	Delete character
`KEY_IC`	Insert char or enter insert mode
`KEY_EIC`	Exit insert char mode
`KEY_CLEAR`	Clear screen
`KEY_EOS`	Clear to end of screen
`KEY_EOL`	Clear to end of line
`KEY_SF`	Scroll 1 line forward
`KEY_SR`	Scroll 1 line backwards

Table 4-6. Constant Values for Function Keys (continued)

Constant	Description
KEY_NPAGE	Next page
KEY_PPAGE	Previous page
KEY_STAB	Set tab
KEY_CTAB	Clear tab
KEY_CATAB	Clear all tabs
KEY_ENTER	Enter or send
KEY_SRESET	Soft (partial) reset
KEY_RESET	Reset or hard reset
KEY_PRINT	Print or copy
KEY_LL	Home down or bottom (lower left)
KEY_A1	Upper left of keypad
KEY_A3	Upper right of keypad
KEY_B2	Center of keypad
KEY_C1	Lower left of keypad
KEY_C3	Lower right of keypad
KEY_BTAB	Back tab
KEY_BEG	Beginning key
KEY_CANCEL	Cancel key
KEY_CLOSE	Close key
KEY_COMMAND	Cmd (command) key
KEY_COPY	Copy key
KEY_CREATE	Create key

Table 4-6. Constant Values for Function Keys (continued)

Constant	Description
KEY_END	End key
KEY_EXIT	Exit key
KEY_FIND	Find key
KEY_HELP	Help key
KEY_MARK	Mark key
KEY_MESSAGE	Message key
KEY_MOVE	Move key
KEY_NEXT	Next object key
KEY_OPEN	Open key
KEY_OPTIONS	Options key
KEY_PREVIOUS	Previous object key
KEY_REDO	Redo key
KEY_REFERENCE	Ref(erence) key
KEY_REFRESH	Refresh key
KEY_REPLACE	Replace key
KEY_RESTART	Restart key
KEY_RESUME	Resume key
KEY_SAVE	Save key
KEY_SBEG	Shifted beginning key
KEY_SCANCEL	Shifted cancel key
KEY_SCOMMAND	Shifted command key
KEY_SCOPY	Shifted copy key

Table 4-6. Constant Values for Function Keys (continued)

Constant	Description
KEY_SCREATE	Shifted create key
KEY_SDC	Shifted delete char key
KEY_SDL	Shifted delete line key
KEY_SELECT	Select key
KEY_SEND	Shifted end key
KEY_SEOL	Shifted clear line key
KEY_SEXIT	Shifted exit key
KEY_SFIND	Shifted find key
KEY_SHELP	Shifted help key
KEY_SHOME	Shifted home key
KEY_SIC	Shifted input key
KEY_SLEFT	Shifted left key
KEY_SMESSAGES	Shifted messages key
KEY_SMOVE	Shifted move key
KEY_SNEXT	Shifted next key
KEY_SOPTIONS	Shifted options key
KEY_SPREVIOUS	Shifted previous key
KEY_SPRINT	Shifted print key
KEY_SREDO	Shifted redo key
KEY_SREPLACE	Shifted replace key

Table 4-6. Constant Values for Function Keys (continued)

Constant	Description
`KEY_SRIGHT`	Shifted right key
`KEY_SRSUME`	Shifted resume key
`KEY_SSAVE`	Shifted save key
`KEY_SSUSPEND`	Shifted suspend key
`KEY_SUNDO`	Shifted undo key
`KEY_SUSPEND`	Suspend key
`KEY_UNDO`	Undo key

The `ungetch()` routine delays processing of *ch* until the next call to `getch()`.

Note The `getch()`, `mvgetch()`, and `mvwgetch()` routines are macros.

Implementation Considerations

The `getch()`, `mvgetch()`, `mvwgetch()`, and `wgetch()` routines are identical to XPG/3. The `ungetch()` routine is a UNIX System V implementation.

See Also

`cbreak()`, `echo()`, `keypad()`, `halfdelay()`, `nodelay()`, `notimeout()`, `raw()`, `wtimeout()`

Portability

The `getch()`, `mvgetch()`, `mvwgetch()`, and `wgetch()` routines conform to HP-UX, UNIX System V, and XPG/3. The `ungetch()` routine conforms to UNIX System V.

getstr
wgetstr
wgetnstr
mvgetstr
mvwgetstr

The `getstr` set of routines is used to get a character string from keyboard.

Syntax

```
#include <curses.h>

int getstr(char *str);
int wgetstr(WINDOW *win, char *str);

int wgetnstr(WINDOW *win, char *str, int n);

int mvgetstr(int y, int x, char *str);
int mvwgetch(WINDOW *win, int y, int x, char *str);
```

Parameters

n	The maximum number of characters to read from input.
str	A pointer to the area where the character string is to be placed.
x	The *x* (column) coordinate of starting position of character string to be read.
y	The *y* (row) coordinate of starting position of character string to be read.
win	A pointer to the window associated with the terminal from which the character is to be read.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `getstr()` and `wgetstr()` routines get a character string from the terminal associated with the window `stdscr` or window *win*, respectively. The `mvgetch()` and `mvwgetch()` routines move the cursor to the position specified in `stdscr` or *win*, respectively, then get a character.

These routines call `getch()` for each character until a newline or carriage return is received, at which time, the string is placed in *str*. The erase and kill characters set by the user are interpreted.

The `wgetnstr()` routine reads at most *n* characters. This routine is used to prevent overflowing the input buffer.

Note The `getstr()`, `mvgetstr()`, `mvwgetstr()` and `wgetstr()` routines are macros.

Implementation Considerations

The `getstr()`, `mvgetstr()`, `mvwgetstr()`, and `wgetstr()` routines are identical to XPG/3. The `wgetnstr()` routine is a UNIX System V implementation.

See Also

`wgetch()`

Portability

The `getstr()`, `mvgetstr()`, `mvwgetstr()`, and `wgetstr()` routines conform to HP-UX, UNIX System V, and XPG/3. The `wgetnstr()` routine conforms to UNIX System V.

getyx
getparyx
getbegyx
getmaxyx

The `getyx` set of routines is used to get positional information for a window.

Syntax

```
#include <curses.h>

void getyx(WINDOW *win, int y, int x);
void getparyx(WINDOW *win, int y, int x);
void getbegyx(WINDOW *win, int y, int x);
void getmaxyx(WINDOW *win, int y, int x);
```

Parameters

win	A pointer to the window from which to get positional information.
x	The integer in which to place x coordinate position of cursor.
y	The integer in which to place y coordinate position of cursor.

Return Values

None

Description

The `getyx()` routine returns the x and y coordinates of the cursor in *win*. The `getparyx()` routine returns the beginning coordinates of *win* relative to its parent window. If *win* is not a subwindow, `getparyx()` sets x and y to -1. The `getbegyx()` routine returns the beginning coordinates of *win* relative to the screen. The `getmaxyx()` routine returns the size of window *win*.

Note These routines are all macros. An ampersand (&) before the *y* and *x* variables is not necessary.

Implementation Considerations

The `getyx()` routine is identical to XPG/3. The `getparyx()`, `getbegyx()`, and `getmaxyx()` routines are UNIX System V implementations.

Portability

The `getyx()` routine conforms to HP-UX, UNIX System V, and XPG/3. The `getparyx()`, `getbegyx()`, and `getmaxyx()` routines conform to UNIX System V.

halfdelay

The `halfdelay` routine is used to enable and disable the half-delay mode.

Syntax

```
#include <curses.h>

int halfdelay(int tenths);
```

Parameters

tenths	The number of tenths of seconds for which to block input (1 to 255).

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `halfdelay()` routine is similar to `cbreak()` in that when set, characters typed by the user are immediately processed by the program. The difference is that `ERR` is returned if no input is received after *tenths* tenths seconds.

The `nocbreak()` routine should be used to leave the half-delay mode.

Implementation Considerations

UNIX System V implementation

See Also

`cbreak()`, `wgetch()`

Portability

UNIX System V

has_color
can_change_color
color_content
pair_content

The `has_color` set of routines is used to get information about colors on terminal.

Syntax

```
#include <curses.h>

bool has_colors();
bool can_change_color();

int pair_content(short pair, short *fg, short *bg);
int color_content(short color, short *r, short *g, short *b);
```

Parameters

color	The number of the color for which to provide information (0 to COLORS).
pair	The number of the color pair for which to provide information (1 to `COLOR_PAIRS` - 1).
r	A pointer to the RGB value for the amount of red in color.
g	A pointer to the RGB value for the amount of green in color.
b	A pointer to the RGB value for the amount of blue in color.
bg	A pointer to the number of the background color (0 to `COLORS`) in pair.
fg	A pointer to the number of the foreground color (0 to `COLORS`) in pair.

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The `has_colors()` routine returns `TRUE` if the terminal supports color. The `can_change_color()` routine returns `TRUE` if the terminal can support color and the colors can be changed. These routines are useful when writing terminal-independent programs; these routines could be used to determine whether to replace color with another attribute on a particular terminal.

The `color_content()` routine provides information on the amount of red, blue, and green in a particular color. The intensity of each color is stored in the addresses pointed to by *r*, *b*, and *g*, respectively. The values returned range from 0 (no component of that color) to 1000 (maximum amount of component).

The `pair_content()` routine provides information on what colors are in the specified color pair. The number of the foreground and background colors are stored in the addresses pointed to by *fg* and *bg*, respectively. The values stored in *fg* and *bg* range from 0 to `COLORS`. The color pair number, *pair*, ranges from 1 to `COLOR_PAIRS -1`.

Implementation Considerations

UNIX System V implementation

See Also

`init_color()`, `init_pair()`, `start_color()`

Portability

UNIX System V

has_ic
has_il

The has_ic and has_il routines are used to determine insert and delete a character or line capability.

Syntax

```
#include <curses.h>

bool has_ic();
bool has_il();
```

Return Values

TRUE	Terminal has insert and delete capability.
FALSE	Terminal does not have insert and delete capability.

Description

The has_ic() routine returns TRUE if the terminal has insert and delete character capability, and FALSE otherwise. Similarly, has_il() returns TRUE if the terminal has insert and delete line capability, and FALSE otherwise.

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

idlok

The `idlok` routine is used to enable the insert and delete line capability.

Syntax

```
#include <curses.h>

int idlok (WINDOW *win, bool bf);
```

Parameters

bf	A Boolean expression.
win	A pointer to the window in which to enable the insert and delete line capability.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `idlok()` routine enables (*bf* is `TRUE`) or disables (*bf* is `FALSE`) the use of the insert and delete line capability of the terminal. By default, the use of insert and delete line is disabled because its use is undesirable for most applications. (Sscreen editor applications are one exception.) When disabled, `CURSES` redraws the changed portions of all lines.

Implementation Considerations

Identical to XPG/3

See Also

`doupdate()`, `scroll()`, `wscrl()`

Portability

HP-UX, UNIX System V, XPG/3

immedok

The `immedok` routine is used to call `wrefresh()` on changes to window.

Syntax

```
#include <curses.h>

int immedok(WINDOW *win, bool bf);
```

Parameters

win	A pointer to the window that is to be refreshed.
bf	A Boolean expression.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

If *bf* is `TRUE`, `immedok()` calls `wrefresh()` if any change to the window image is made (for example, through routines such as `addch()`, `wclrtobot()`, and `wscrl()`). Repeated calls to `wrefresh()` may affect performance negatively. The `immedok()` routine is disabled by default.

Implementation Considerations

UNIX System V implementation

See Also

`waddch()`, `wclrtobot()`, `wrefresh()`, `wscrl()`

Portability

UNIX System V

inch
winch
mvinch
mvwinch

The `inch` set of routines is used to return a character (with attributes).

Syntax

```
#include <curses.h>

chtype inch;
chtype winch(WINDOW *win);

chtype mvinch(int y, int x);
chtype mvwinch(WINDOW *win, int y, int x);
```

Parameters

ch	The character to be returned.
win	A pointer to the window that contains the character to be returned.
x	The *x* (column) coordinate of the position of the character to be returned.
y	The *y* (row) coordinate of the position of the character to be returned.

Return Values

The `chtype ()` character from the screen location.

Description

The `inch()` and `winch()` routines return the `chtype` character located at the current cursor position of the `stdscr` window and window *win*, respectively. The `mvinch()` and `mvwinch()` routines return the `chtype` character located at the position indicated by the *x* (column) and *y* (row) parameters (the former in the `stdscr` window; the latter in window *win*).

The complete character and attribute pair is returned. The character or attributes can be extracted by performing a bitwise `AND` on the returned value, using the constants `A_CHARTEXT`, `A_ATTRIBUTES`, and `A_COLOR` defined in `curses.h`.

Note All of these routines are macros.

Implementation Considerations

Identical to XPG/3

See Also

`waddch()`, `wattrset()`

Portability

UNIX System V, XPG/3

inchstr
winchstr
inchnstr
winchnstr
mvinchstr
mvwinchstr
mvinchnstr
mvwinchnstr

The `inchstr` set of routines is used to return a character string (with attributes).

Syntax

```
#include <curses.h>

int inchstr(chtype *chstr);
int winchstr(WINDOW *win, chtype *chstr);

int inchnstr(chtype *chstr, int n);
int winchnstr(WINDOW *win, chtype *chstr, int n);

int mvinchstr(int y, int x, chtype *chstr);
int mvwinchstr(WINDOW *win, int y, int x, chtype *chstr);

int mvinchnstr(int y, int x, chtype *chstr, int n);
int mvwinchnstr(WINDOW *win, int y, int x, chtype *chstr, int n);
```

Parameters

n	The number of characters not to exceed when returning *chstr*.
chstr	The character string to be returned.
win	A pointer to the window in which the string is to be returned.

x The *x* (column) coordinate of the starting position of the string to be returned.

y The *y* (row) coordinate of the starting position of the string to be returned.

Return Values

`OK` Successful completion.

`ERR` An error occurred.

Description

The `inchstr()` and `winchstr()` routines return the character string (with attributes) starting at the current cursor position of the `stdscr` window and window *win*, respectively, and ending at the right margin. The `mvinchstr()` and `mvwinchstr()` routines return the character string located at the position indicated by the *x* (column) and *y* (row) parameters (the former in the `stdscr` window; the latter in window *win*).

The `inchnstr()`, `winchnstr()`, `mvinchnstr()`, and `mvwinchnstr()` routines return at most *n* characters from the window `stdscr` and *win*, respectively. The former two routines return the string, starting at the current cursor position; the latter two commands return the string, starting at the position specified by the *x* and *y* parameters.

The complete character/attribute pair is returned. The character or attributes can be extracted by performing a bitwise AND on the returned value, using the constants `A_CHARTEXT`, `A_ATTRIBUTES`, and `A_COLOR` defined in `curses.h`. The character string can also be returned without attributes by using `winstr()`.

Note All routines except `winchnstr()` are macros.

Implementation Considerations

UNIX System V implementation

See Also

`winch()`, `winstr()`

Portability

UNIX System V

init_color
init_pair

The `init_color` and `init_pair` routines are used to initialize a color pair.

Note The `init_color` and `init_pair` routines are not implemented at this time.

Syntax

```
#include <curses.h>

int init_color(short color, short r, short g, short b);
int init_pair(short pair, short fg, short bg);
```

Parameters

color	The number of the color to be changed (0 to `COLORS`).
pair	The number of the color pair to be changed (1 to `COLOR_PAIRS -1`).
r	The RGB value for the amount of red in color (0 to 1000).
g	The RGB value for the amount of green in color (0 to 1000).
b	The RGB value for the amount of blue in color (0 to 1000).
bg	The number of the background color (0 to `COLORS`).
fg	The number of the foreground color (0 to `COLORS`).

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `init_pair()` routine initializes a color pair so that the macro `COLOR_PAIR(`*n*`)` can be used as an attribute. Its first argument is the number of the color pair to be changed; the second argument is the number of the foreground color; the third argument is the number of the background color. The maximum number of color pairs and colors that the terminal can support are defined in the global variables `COLOR_PAIRS` and `COLORS`, respectively.

Each time that a color pair is initialized, the screen is refreshed and all occurrences of that color pair are updated to reflect the new definition.

The `init_color()` routine redefines the color using the number of the color and the RGB values for red, blue, and green as arguments.

The following default colors are defined in `curses.h`. (`CURSES` assumes that `COLOR_BLACK`) is the default background color for all terminals.)

`COLOR_BLACK`

`COLOR_RED`

`COLOR_GREEN`

`COLOR_YELLOW`

`COLOR_BLUE`

`COLOR_MAGENTA`

`COLOR_CYAN`

`COLOR_WHITE`

Each time that a color is redefined with `init_color()`, the screen is refreshed, and all occurrences of that color are updated to reflect the new definition.

Implementation Considerations

UNIX System V implementation

See Also

can_change_color(), color_content(), has_color(), pair_content(), start_color()

Portability

UNIX System V

initscr

The `initscr` routine is used to initialize single terminal environment.

Syntax

```
#include <curses.h>

WINDOW *initscr();
```

Return Values

On success, a pointer to `stdscr` is returned; otherwise, a null pointer is returned (for example, if the console could not be opened for write; the terminal could not be initialized; or memory could not be allocated for `stdscr`).

Description

The `initscr()` routine initializes `CURSES` data structures, determines the terminal type, and makes sure that the first call to `refresh()` clears the screen. If the program interacts with only one terminal, this should be the first routine called.

If the program interacts with more than one terminal, `newterm()` should be called for each terminal instead of a single call to `initscr()`.

Implementation Considerations

Identical to XPG/3

See Also

`endwin()`, `is_endwin()`, `newterm()`, `set_term()`, `use_env()`

Portability

HP-UX, UNIX System V, XPG/3

insch
winsch
mvinsch
mvwinsch

The `insch` set of routines is used to insert a character.

Syntax

```
#include <curses.h>

int insch(chtype ch);
int winsch(WINDOW *win, chtype ch);

int mvinsch(int y, int x, chtype ch);
int mvwinsch(WINDOW *win, int y, int x, chtype ch);
```

Parameters

ch	The character to be inserted.
win	A pointer to the window in which the character is to be inserted.
x	The *x* (column) coordinate of the position of the character.
y	The *y* (row) coordinate of the position of the character.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `insch()` routine inserts the `chtype` character at the current cursor position of the `stdscr` window. The `winsch()` routine performs the identical action but in window *win*. The `mvinsch()` and `mvwinsch()` routines insert the character at the position indicated by the x (column) and y (row) parameters (the former in the `stdscr` window; the latter in window *win*). The cursor position does not change.

All characters to the right of the inserted character are moved right one character. The last character on the line is deleted.

Note All routines except `winsch()` are macros.

Implementation Considerations

Identical to XPG/3

See Also

`delch()`, `insstr()`

Portability

HP-UX, UNIX System V, XPG/3

insdelln
winsdelln

The `insdelln` and `winsdelln` routines are used to insert or delete lines to or from the window.

Syntax

```
#include <curses.h>

int insdelln(int n);
int winsdelln(WINDOW *win, int n);
```

Parameters

win	A pointer to the window in which to insert or delete a line.
n	The number of lines to insert or delete (positive *n* inserts; negative *n* deletes).

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `insdelln()` and `winsdelln()` routines insert or delete blank lines in `stdscr` or *win*, respectively.

When *n* is positive, *n* lines are added above the current line, and the bottom *n* lines are cleared; when *n* is negative, *n* lines are deleted starting with the current line, and the remaining lines are moved up. The bottom *n* lines are cleared. The position of the cursor does not change.

Implementation Considerations

UNIX System V implementation

See Also

`wdeleteln()`, `winsertln()`

Portability

UNIX System V

insertln
winsertln

The `insertln` and `winsertln` routines are used to insert a line in a window.

Syntax

```
#include <curses.h>

int insertln();
int winsertln(WINDOW *win);
```

Parameters

win	A pointer to the window in which to insert the line.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `insertln()` and `winsertln()` routines insert a blank line above the current line in `stdscr` or *win*, respectively. The new line becomes the current line. All lines below the current line in the window are moved down one line. The bottom line in the window is discarded.

Note These routines are macros.

Implementation Considerations

Identical to XPG/3

See Also

`wbkgdset()`, `wdeleteln()`, `winsdelln()`

Portability

HP-UX, UNIX System V, XPG/3

insstr
winsstr
insnstr
winsnstr
mvinsstr
mvwinsstr
mvinsnstr
mvwinsnstr

The `insstr` set of routines is used to insert a character string.

Syntax

```
#include <curses.h>

int insstr(char *str);
int winsstr(WINDOW *win, char *str);

int insnstr(char *str, int n);
int winsnstr(WINDOW *win, char *str, int n);

int mvinsstr(int y, int x, char *str);
int mvwinsstr(WINDOW *win, int y, int x, char *str);

int mvinsnstr(int y, int x, char *str, int n);
int mvwinsnstr(WINDOW *win, int y, int x, char *str, int n);
```

Parameters

str	A pointer to the string to be inserted.
n	The number of characters not to exceed when inserting *str*. If *n* is less than 0, the entire string is inserted.
win	A pointer to the window in which the string is to be inserted.
x	The *x* (column) coordinate of the starting position of the string.

y The *y* (row) coordinate of the starting position of the string.

Return Values

`OK` Successful completion.

`ERR` An error occurred.

Description

The `insstr()` routine inserts *str* at the current cursor position of the `stdscr` window. The `winsstr()` routine performs the identical action, but in window *win*. The `mvinsstr()` and `mvwinsstr()` routines insert the character string at the starting position indicated by the *x* (column) and *y* (row) parameters (the former to the `stdscr` window; the latter to window *win*).

The `insnstr()`, `winsnstr()`, `mvinsnstr()`, and `mvwinsnstr()` routines insert *n* characters to the window or as many as will fit on the line. If *n* is less than 0, the entire string is inserted, or as much of it as fits on the line. The former two routines place the string at the current cursor position; the latter two commands use the position specified by the *x* and *y* parameters.

All characters to the right of inserted characters are moved to the right. Characters that do not fit on the current line are discarded. The logical cursor is left at the point of insertion.

If a character in *str* is a newline, carriage return, backspace, or tab, the cursor is moved appropriately. The cursor is moved to the next tab stop for each tab character (tabs are eight characters apart). If the character is a control character other than those previously mentioned, the character is inserted using ^x notation, where *x* is a printable character. The `clrtoeol()` routine is automatically done before a newline.

Note All routines except `winsnstr()` are macros.

Implementation Considerations

UNIX System V implementation

See Also

`waddstr()`, `winsch()`

Portability

UNIX System V

instr
winstr
innstr
winnstr
mvinstr
mvwinstr
mvinnstr
mvwinnstr

The `instr` set of routines is used to return a character string (without attributes).

Syntax

```
#include <curses.h>

int instr(char *str);
int winstr(WINDOW *win, char *str);

int innstr(char *str, int n);
int winnstr(WINDOW *win, char *str, int n);

int mvinstr(int y, int x, char *str);
int mvwinstr(WINDOW *win, int y, int x, char *str);

int mvinnstr(int y, int x, char *str, int n);
int mvwinnstr(WINDOW *win, int y, int x, char *str, int n);
```

Parameters

n	The number of characters not to exceed when returning *str*.
str	A character string to be returned.
win	A pointer to the window in which the string is to be returned.
x	The x (column) coordinate of the starting position of the string to be returned.

y The *y* (row) coordinate of the starting position of the string to be returned.

Return Values

OK Successful completion.

ERR An error occurred.

Description

The `instr()` and `winstr()` routines return the character string (without attributes) starting at the current cursor position of the `stdscr` window and window *win*, respectively, and ending at the right margin. The `mvinstr()` and `mvwinstr()` routines return the character string located at the position indicated by the *x* (column) and *y* (row) parameters (the former in the `stdscr` window; the latter in window *win*).

The `innstr()`, `winnstr()`, `mvinnstr()`, and `mvwinnstr()` routines return at most *n* characters from the window `stdscr` and *win*, respectively. The former two routines return the string starting at the current cursor position; the latter two commands return the string, starting at the position specified by the *x* and *y* parameters.

Only the character portion of the character/attribute pair is returned. To return the complete character/attribute pair, use `winchstr()`.

Note All routines except `winnstr()` are macros.

Implementation Considerations

UNIX System V implementation

See Also

`winch()`, `winchstr()`

Portability

UNIX System V

intrflush

The `intrflush` routine is used to flush output in tty on interrupt.

Syntax

```
#include <curses.h>

int intrflush(WINDOW *win, bool bf);
```

Parameters

bf	A Boolean expression.
win	An ignored parameter.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

If this option is enabled (*bf* is `TRUE`), `intrflush()` flushes all output in the terminal driver when an interrupt, quit, or suspend character is sent to the terminal. This increases interrupt reponse time but causes `CURSES` to lose track of what currently exists on the screen. Whether this option is enabled or disabled by default depends on the tty driver.

Note The `intrflush()` routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`flushinp()`, `qiflush()`, `noqiflush()`

Portability

HP-UX, UNIX System V, XPG/3

keyname

The `keyname` routine is used to return the character string for a key.

Syntax

```
#include <curses.h>

char *keyname(int c);
```

Parameters

c	The key for which to get the name.

Return Values

None

Description

The `keyname()` routine returns a string pointer to the key name. Make a duplicate of the returned string if you plan to modify it.

Implementation Considerations

UNIX System V implementation

Portability

UNIX System V

keypad

The `keypad` routine is used to enable keypad handling.

Syntax

```
#include <curses.h>

int keypad(WINDOW *win, bool bf);
```

Parameters

win	A pointer to the window in which to enable keypad handling.
bf	A Boolean expression.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

If *bf* is `TRUE`, `keypad()` handles special keys from the keyboard on the terminal associated with *win* as single values instead of character sequences. For example, if the user presses the right arrow key, `wgetch()` returns a single value, `KEY_RIGHT`, that represents the function key; otherwise, `CURSES` handles the special keys as normal text.

See `wgetch()` for a list of tokens for function keys that are returned by `getch()`.

Implementation Considerations

Identical to XPG/3

See Also

`wgetch()`

Portability

HP-UX, UNIX System V, XPG/3

killchar

The `killchar` routine is used to return the current `KILL` character.

Syntax

```
#include <curses.h>

char killchar();
```

Return Values

The terminal's current `KILL` character is returned.

Description

The `killchar()` routine returns the user's choice of `KILL` character from the tty driver. This character is used to start a new line of input when the current input is considered erroneous. The returned value can be used when including deletion capability in interactive programs.

Implementation Considerations

Identical to XPG/3

See Also

`erasechar()`, `wgetnstr()`

Portability

HP-UX, UNIX System V, XPG/3

leaveok

The `leaveok` routine is used to ignore cursor relocation.

Syntax

```
#include <curses.h>

int leaveok(WINDOW *win, bool bf);
```

Parameters

win	A pointer to the window in which to ignore the position of the cursor.
bf	A Boolean expression.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

If *bf* is `TRUE`, `leaveok()` leaves the cursor in a position that `CURSES` finds convenient at the time that the window is refreshed. Normally, when a window is refreshed, `leaveok()` is disabled and the cursor is mapped from the logical window to the same location in the physical window.

Enabling `leavok()` is useful when the cursor is not used or is not important in the application. Reducing cursor movements simplifies program interaction.

Once `leaveok()` is set to `TRUE`, it remains enabled until another call sets it to `FALSE` or until the program terminates.

Implementation Considerations

Identical to XPG/3

See Also

`wrefresh()`

Portability

HP-UX, UNIX System V, XPG/3

longname

The `longname` routine is used to return the full terminal type name.

Syntax

```
#include <curses.h>

char *longname();
```

Return Values

On success, pointer to verbose description of terminal is returned; otherwise, a null pointer is returned.

Description

The `longname()` routine returns a pointer to a static area containing a verbose description (128 characters or less) of the terminal. The area is defined after calls to `initscr()`, `newterm()`, or `setupterm()`. The value should be saved if `longname()` is going to be used with multiple terminals since it is overwritten with a new value after each call to `setupterm()`.

Implementation Considerations

Identical to XPG/3

See Also

`initscr()`, `newterm()`, `setupterm()`

Portability

HP-UX, UNIX System V, XPG/3

meta

The `meta` routine is used to control the number of bits returned on input.

Syntax

```
#include <curses.h>

int meta(WINDOW *win, bool bf);
```

Parameters

bf	A Boolean expression.
win	An ignored parameter.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred. The terminal or system cannot handle 8-bit character codes.

Description

Whether a terminal returns 7 or 8 significant bits initially depends on the control mode of the terminal driver. The `meta()` routine forces the number of bits to be returned by `getch()` to be 7 (if *bf* is `FALSE`) or 8 (if *bf* is `TRUE`).

Note If the program handling the data can pass only 7-bit characters or strips the 8th bit, 8 bits cannot be handled.

If the `terminfo` capabilities `smm` (`meta_on`) and `rmm` (`meta_off`) are defined for the terminal, `smm` is sent to the terminal when meta(*win*, `TRUE`) is called, and `rmm` is sent when meta(*win*, `FALSE`) is called.

This routine is useful when extending the nontext command set in applications where the `META` key is used.

Note The `meta()` routine is provided for compatability with older `CURSES` packages. The MPE/iX `CURSES` package handles 16-bit characters, therefore making this function unnecessary.

Implementation Considerations

HP-UX and UNIX System V implementations

Portability

HP-UX, UNIX System V

move
wmove

The `move` and `wmove` routines are used to move the cursor in the window.

Syntax

```
#include <curses.h>

int move(int y, int x);
int wmove(WINDOW *win, int y, int x);
```

Parameters

win	A pointer to the window in which the cursor is to be written.
x	The *x* (column) coordinate of the position of the cursor in the window.
y	The *y* (row) coordinate of the position of the cursor in the window.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred. The cursor is outside the window boundary.

Description

The `move()` routine moves the logical cursor (for `stdscr`) to the position specified by *y* (row) and *x* (column), where the upper-left corner of the window is row 0, column 0. The `wmove()` routine performs the same action, but moves the cursor in the window specified by *win*. The physical cursor does not move until after a call to `wrefresh()` or `doupdate()`.

Note The `move()` routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`doupdate()`, `wrefresh()`

Portability

HP-UX, UNIX System V, XPG/3

mvcur

The `mvcur` routine is used to move the cursor (interface to `terminfo`).

Syntax

```
#include <curses.h>

int mvcur (int oldrow, int oldcol, int newrow, int newcol);
```

Parameters

oldrow The row from which cursor is to be moved.

oldcol The column from which cursor is to be moved.

newrow The row to which cursor is to be moved.

newcol The column to which cursor is to be moved.

Return Values

`OK` Successful completion.

`ERR` An error occurred.

Description

The `mvcur()` routine is a low-level routine used only outside of `CURSES` when the program has to deal directly with the database to handle certain terminal capabilities. The use of appropriate `CURSES` routines is recommended in all other situations.

The `mvcur()` routine moves the cursor from the location specified by *oldrow* and *oldcol* to the location specified by *newrow* and *newcol*. The program must keep track of the current cursor position. All output will be sent to `stdout` through `_putchar()`

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V

mvwin

The `mvwin` routine is used to move a window.

Syntax

```
#include <curses.h>

int mvwin(WINDOW *win, int y, int x);
```

Parameters

win	A pointer to the window to move.
y	The *y* (row) coordinate of the upper-left corner of the window.
x	is the *x* (column) coordinate of the upper-left corner of the window.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred. The move places part of or all of window outside the screen boundary (or, in the case of a subwindow, outside its parent window's boundaries.)

Description

The `mvwin()` routine moves the specified window (or subwindow), placing its upper left corner at the positions specified by *x* and *y*. The entire window must fit within the physical boundaries of the screen, or an error results. A subwindow must fit within the boundaries of its parent window.

Implementation Considerations

Identical to XPG/3

See Also

`newwin()`, `subwin()`

Portability

HP-UX, UNIX System V, XPG/3

napms

Note The `napms()` routine is not implemented at this time.

newpad

The `newpad` routine is used to create a new pad.

Syntax

```
#include <curses.h>

WINDOW *newpad(int nlines, int ncols);
```

Parameters

nlines	The number of lines in the window.
ncols	The number of columns in the window.

Return Values

On success, a pointer to the new window structure is returned; otherwise, a null pointer is returned.

Description

The `newpad()` routine creates a new pad with the specified number of lines and columns. A pointer to the new pad structure is returned. A pad differs from a window in that it is not restricted to the size of the physical screen. It is useful when only part of a large window will be displayed at any one time.

Automatic refreshes by scrolling or echoing of input do not take place when pads are used. Pads have their own refresh commands, `prefresh()` and `pnoutrefresh()`. These contain additional parameters for specifying what part of the pad to display and where to display it on the screen.

Implementation Considerations

Identical to XPG/3

See Also

`pnoutrefresh()`, `prefresh()`

Portability

HP-UX, UNIX System V, XPG/3

newterm

The `newterm` routine is used to open a new terminal.

Syntax

```
#include <stdio.h>
#include <curses.h>
SCREEN *newterm(char *type, FILE *outfp, FILE *infp);
```

Parameters

type	A string defining the terminal type to be used in place of `TERM`.
outfp	A pointer to a file to be used for output to the terminal.
infp	The pointer to a file to be used for input to the terminal.

Return Values

On success, a pointer to new `SCREEN` structure is returned; otherwise, a null pointer is returned.

Description

The `newterm()` routine opens a new terminal with each call. It should be used instead of `initscr()` when the program interacts with more than one terminal. It returns a variable of type `SCREEN`, which should be used for later reference to that terminal. Before program termination, `endwin()` should be called for each terminal.

Implementation Considerations

Identical to XPG/3

See Also

`delscreen()`, `endwin()`, `initscr()`

Portability

HP-UX, UNIX System V, XPG/3

newwin

The `newwin` routine is used to create a window.

Syntax

```
#include <curses.h>

WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x);
```

Parameters

nlines	The number of lines in the new window.
ncols	The number of columns in the new window.
begin_y	The y (row) coordinate of the position of the upper left corner of window.
begin_x	The x (column) coordinate of the position of the upper left corner of the window.

Return Values

On success, pointer to new window structure is returned; otherwise, a null pointer is returned.

Description

The `newwin()` routine creates a new window with the specified number of lines and columns and upper left corner positioned at *begin_x*, *begin_y*. A pointer to the new window structure is returned. A full-screen window can be created by calling `newwin(0,0,0,0)`.

If the number of lines specified is zero, `newwin()` uses the default `LINES` minus *begin_y*; if the number of columns specified is zero, `newwin()` uses the default `COLS` minus *begin_x*.

Note The `newwin()` routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`delwin()`, `derwin()`, `dupwin()`, `mvwin()`, `subwin()`, `touchwin()`

Portability

HP-UX, UNIX System V, XPG/3

nl
nonl

The `nl` and `nonl` routines are used to enable and disable newline control.

Syntax

```
#include <curses.h>

int nl();
int nonl();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `nl()` routine enables the handling of newlines. The `nl()` routine converts newline into carriage return and line feed on output and converts carriage return into newline on input. The `nonl()` routine disables the handling of newlines.

The handling of newlines is initially enabled. Disabling the handling of newlines results in faster cursor motion since `CURSES` can use the line-feed capability more efficiently.

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

nodelay

The `nodelay` routine is used to set blocking or non-blocking read.

Syntax

```
#include <curses.h>

int nodelay(WINDOW *win, bool bf);
```

Parameters

bf	A Boolean expression.
win	A pointer to the window in which to enable non-blocking.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

If enabled (*bf* is `TRUE`), `nodelay()` causes `getch()` to return `ERR` if no input is ready. When disabled, `getch()` blocks until a key is pressed.

Note The `nodelay()` routine is a macro.

Implementation Considerations

Identical to XPG/3

See Also

`wgetch()`, `wtimeout()`

Portability

HP-UX, UNIX System V, XPG/3

notimeout

The `notimeout` routine is used to disable the timer used by `getch()`.

Syntax

```
#include <curses.h>

int notimeout(WINDOW *win, bool bf);
```

Parameters

bf	A Boolean expression.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

If `bool` is `TRUE`, `notimeout()` disables a timer used by `getch()` when interpreting escape character sequences.

When `bool` is `FALSE` and keypad handling is enabled, a timer is set by `wgetch()` to handle characters received that could be the beginning of a function key (for example, ESC). If the remainder of the sequence is not received before the time expires, the character is returned; otherwise, the value of the function key is returned. If `notimeout()` is set to `TRUE`, the timer is not set and all characters are returned as single values.

Note The `notimeout()` routine is a macro.

Implementation Considerations

UNIX System V implementation

See Also

`keypad()`, `wgetch()`

Portability

UNIX System V

overlay
overwrite

The `overlay` and `overwrite` routines are used to overlap or overwrite windows.

Syntax

```
#include <curses.h>

int overlay(WINDOW *srcwin, WINDOW *dstwin);
int overwrite(WINDOW *srcwin, WINDOW *dstwin);
```

Parameters

srcwin	A pointer to the source window to be copied.
dstwin	A pointer to the destination window to be overlayed or overwritten.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `overwrite()` and `overlay()` routines copy *srcwin* to *destwin*. The source window (*srcwin*) and destination window (*dstwin*) do not have to be the same size.

The `overwrite()` routine copies all characters to *dstwin*; thus, destroying all previous contents of the window. The `overlay()` routine copies only nonblank characters, leaving blank characters intact. Thus, if the background character of the original window was set to something other than a blank, this original background could be preserved.

The example shown on the following pages illustrates how to use `overwrite()` to implement a pop-up dialog box.

```
#include <curses.h>

/*
 *   Pop-up a window on top of curscr.  If row and/or col
 *   are -1 then that dimension will be centered within
 *   curscr.  Return 0 for success or -1 if malloc() failed.
 *   Pass back the working window and the saved window for the
 *   pop-up.  The saved window should not be modified.
 */
int
popup(work, save, nrows, ncols, row, col)
WINDOW **work, **save;
int nrows, ncols, row, col;
{
      int mr, mc;
      getmaxyx(curscr, mr, mc);
      /* Windows are limited to the size of curscr. */
      if (mr < nrows)
            nrows = mr;
      if (mc < ncols)
            ncols = mc;
      /* Center dimensions. */
      if (row == -1)
            row = (mr-nrows)/2;
      if (col == -1)
            col = (mc-ncols)/2;
      /* The window must fit entirely in curscr. */
      if (mr < row+nrows)
            row = 0;
      if (mc < col+ncols)
            col = 0;
      *work = newwin(nrows, ncols, row, col);
      if (*work == NULL)
            return (-1);
      if ((*save = dupwin(*work)) == NULL) {
            delwin(*work);
            return (-1);
```

```
     }
     overwrite(curscr, *save);
     return (0);
}

/*
 *   Restore the region covered by a popup window.
 *   Delete the working window and the saved window.


 *   This function is the complement to popup().  Return
 *   0 for success or -1 for an error.
 */
int
popdown(work, save)
WINDOW *work, *save;
{
      (void) overwrite(save, curscr);
      (void) delwin(save);
      (void) delwin(work);
      return (0);
}

/*
 *   Compute the size of a dialog box that would fit around
 *   the string.
 */
void
dialsize(str, nrows, ncols)
char *str;
int *nrows, *ncols;
```

```
{
      int rows, cols, col;
      for (rows = 1, cols = col = 0; *str != ' '; ++str) {
            if (*str == '0) {
                  if (cols < col)
                        cols = col;
                  col = 0;
                  ++rows;
            } else {
                  ++col;
            }
      }
      if (cols < col)
            cols = col;
      *nrows = rows;
      *ncols = cols;
}

/*
 *   Write a string into a dialog box.
 */
void
dialfill(w, s)
WINDOW *w;
char *s;
{
      int row;
      (void) wmove(w, 1, 1);
      for (row = 1; *s != ' '; ++s) {
            (void) waddch(w, *((unsigned char*) s));
            if (*s == '0)
                  wmove(w, ++row, 1);
      }
      box(w, 0, 0);
```

```
}

void
dialog(str)
char *str;
{
     WINDOW *work, *save;
     int nrows, ncols, row, col;

     /* Figure out size of window. */
     dialsize(str, &nrows, &ncols);

     /* Create a centered working window with extra */
     /* room for a border. */
     (void) popup(&work, &save, nrows+2, ncols+2, -1, -1);

     /* Write text into the working window. */
     dialfill(work, str);

     /* Pause.  Remember that wgetch() will do a wrefresh() */
     /* for us. */
     (void) wgetch(work);

     /* Restore curscr and free windows. */
     (void) popdown(work, save);

     /* Redraw curscr to remove window from physical screen. */
     (void) doupdate();
}
```

Implementation Considerations

Identical to XPG/3

See Also

`copywin()`

Portability

HP-UX, UNIX System V, XPG/3

pair_content

Note The `pair_content` routine is not implemented at this time.

prefresh
pnoutrefresh

The `prefresh` and `pnoutrefresh` routines routines are used to copy the pad data structure to a physical window.

Syntax

```
#include <curses.h>

int prefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
   int smincol, int smaxrow, int smaxcol);

int pnoutrefresh(WINDOW *pad, int pminrow, int pmincol, int sminrow,
   int smincol, int smaxrow, int smaxcol);
```

Parameters

pad	A pointer to the pad to refresh.
pmincol	The column coordinate of the upper-left corner of the pad rectangle to be copied.
pminrow	The row coordinate of the upper-left corner of the pad rectangle to be copied
smincol	The column coordinate of the upper-left corner of the rectangle on the physical screen where pad is to be positioned.
sminrow	The row coordinate of the upper-left corner of the rectangle on the physical screen where pad is to be positioned.
smaxcol	The column coordinate of the lower-right corner of the rectangle on the physical screen where the pad is to be positioned.
smaxrow	The row coordinate of the lower-right corner of the rectangle on the physical screen where the pad is to be positioned.

Return Values

`OK`	Successful completion.
`ERROR`	An error occurred.

Description

The `prefresh()` routine copies the specified portion of the logical pad to the terminal screen. The parameters *pmincol* and *pminrow* specify the upper-left corner of the rectangular area of the pad to be displayed. The lower-right coordinate of the rectangular area of the pad that is to be displayed is calculated from the screen parameters (*sminrow*, *smincol*, *smaxrow*, and *smaxcol*).

This routine calls the `pnoutrefresh()` routine to copy the specified portion of pad to the terminal screen and the `doupdate()` routine to do the actual update. The logical cursor is copied to the same location in the physical window unless `leavok()` is enabled (in which case, the cursor is placed in a position that the program finds convenient).

When outputting several pads at once, it is often more efficient to call the `pnoutrefresh()` and `doupdate()` routines directly. A call to `pnoutrefresh()` for each pad first, followed by only one call to `doupdate()` to update the screen, results in one burst of output, fewer characters sent, and less CPU time used.

Implementation Considerations

Identical to XPG/3

See Also

`doupdate()`, `leaveok()`, `newpad()`,

Portability

HP-UX, UNIX System V, XPG/3

printw
wprintw
mvprintw
mvwprintw
vwprintw

The `printw` set of routines is used to perform a formatted write to a window.

Syntax

```
#include <curses.h>

int printw(char *fmt [,arg...]);
int wprintw(WINDOW *win, char *fmt[,arg...]);

int mvprintw(int y, int x, char *fmt [,arg...]);
int mvwprintw(WINDOW *win, int y, int x, char *fmt [,arg...])

#include <stdargs.h>

vwprintw(WINDOW *win, char *fmt, va_list arglist);
```

Parameters

fmt [,arg...]	A `printf()` format string where *arg* is zero or more arguments used to satisfy the `printf()` string.
fmt, *arglist*	A `vprintf()` format string where *arglist* is a pointer to a list of arguments.
win	A pointer to the window in which the string is to be written.
x	The *x* (column) coordinate position of the string's placement in the window.
y	The *y* (row) coordinate position of the string's placement in the window.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

These routines are functionally equivalent to `printf()`. Characters are written to the window using `waddch()`.

With `printw()` and `wprintw()`, the characters are written to `stdscr` and *win*, respectively. The `mvprintw()` and `mvwprintw()` routines position the cursor as specified in `stdscr` or *win*, respectively, and then call `printw()`. The `vwprintw()` routine writes to the window.

Implementation Considerations

The `printw()`, `wprintw()`, `mvprintw()`, and `mvwprintw()` routines are identical to XPG/3. The `vwprintw()` routine is a UNIX System V implementation.

See Also

`waddch()`

Portability

The `printw()`, `wprintw()`, `mvprintw()`, and `mvwprintw()` routines conform to HP-UX, UNIX System V and XPG/3. The `vwprintw()` routine conforms to UNIX System V.

qiflush
noqiflush

The `qiflush` and `noqiflush` set of routines is used to control the flush of input and output on interrupt.

Syntax

```
#include <curses.h>

void qiflush();
void noqiflush();
```

Return Values

None

Description

The `qiflush()` routine flushes input and output queues when an interrupt, quit, or suspend character is sent to the terminal. The `noqiflush()` routine does not flush input and output queues when these characters are sent.

Implementation Considerations

UNIX System V implementation

See Also

`flushinp()`, `intrflush()`

Portability

UNIX System V

raw
noraw

The `raw` and `noraw` routines are used to enable and disable the raw-mode operation.

Syntax

```
#include <curses.h>

int raw();
int noraw();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `raw()` and `noraw()` routines enable and disable raw-mode operation, respectively. These routines are similar to `cbreak()` and `nocbreak()` in that `raw()` immediately processes characters typed by the user, and `noraw()` restores the previous state. The difference is that `raw()` passes through quit, interrupt, suspend, and flow control characters (`QUIT`, `INTR`, `SUSP`, `STOP`, `START`) as normal text without generating a signal. The behavior of the `BREAK` key varies depending on the terminal.

Implementation Considerations

Identical to XPG/3

See Also

`cbreak()`, `wgetch()`, `halfdelay()`, `nodelay()`, `wtimeout()`

Portability

HP-UX, UNIX System V, XPG/3

redrawwin
wredrawln

The `redrawwin` and `wredrawln` routines are used to redraw the screen or portion of the screen.

Syntax

```
#include <curses.h>

int redrawwin(WINDOW *win);
int wredrawln(WINDOW *win, int beg_line, int num_lines);
```

Parameters

win	A pointer to the window in which to redraw.
beg_line	The first line to redraw.
num_lines	The number of lines to redraw.

Return Values

`OK`	Successful completion.
`ERROR`	An error occurred.

Description

The `redrawwin()` and `wredrawln()` routines force portions of a window to be redrawn to the terminal. These routines are useful when the data that exists on the screen is believed to be corrupt and for applications such as screen editors that redraw portions of the screen.

Note The `redrawwin()` routine is a macro.

Implementation Considerations

UNIX System V implementation

Portability

UNIX System V

refresh
wrefresh
doupdate
wnoutrefresh

The `refresh` set of routines is used to copy a window data structure to a physical window.

Syntax

```
#include <curses.h>

int refresh();
int wrefresh(WINDOW *win);

int doupdate();

int wnoutrefresh(WINDOW *win);
```

Parameters

win	A pointer to the window in which to refresh.

Return Values

`OK`	Successful completion.
`ERROR`	An error occurred.

Description

The `refresh()` and `wrefresh()` routines copy `stdscr` and *win*, respectively, to the terminal screen. These routines call the `wnoutrefresh()` routine to copy the specified window to *curscr* and the `doupdate()` routine to do the actual update. The physical cursor is mapped to the same position as the logical cursor of the last window to update *curscr*, unless `leaveok()` is enabled (in which case, the cursor is placed in a position that `CURSES` finds convenient).

When outputting several windows at once, it is often more efficient to call the `wnoutrefresh()` and `doupdate()` routines directly. A call to `wnoutrefresh()`for each window first, followed by only one call to `doupdate()` to update the screen, results in one burst of output, fewer characters sent, and less CPU time used.

If the *win* parameter to `wrefresh()` is global variable *curscr*, the screen is immediately cleared and repainted from scratch.

Implementation Considerations

Identical to XPG/3

See Also

`leaveok()`, `pnoutrefresh()`, `prefresh()`, `redrawln()`, `redrawwin()`

Portability

HP-UX, UNIX System V, XPG/3

reset_prog_mode
reset_shell_mode

The `reset_prog_mode` and `reset_shell_mode` routines are used to reset the terminal modes.

Syntax

```
#include <curses.h>

int reset_prog_mode();
int reset_shell_mode();
```

Return Values

`OK`	Successful completion.

Description

The `reset_prog_mode()` and `reset_shell_mode()` routines reset the current terminal modes to "program" (within `CURSES`) or "shell" (outside `CURSES`). The reset is done automatically by `endwin()` and by `doupdate()` after a call to `endwin()`.

Implementation Considerations

Identical to XPG/3

Portability

UNIX System V, XPG/3

resetty
savetty

The `resetty` and `savetty` routines are used to restore and save terminal modes.

Syntax

```
#include <curses.h>

int resetty();
int savetty();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `savetty()` and `resetty()` routines are low-level routines typically used within library routines. The `savetty()` and `resetty()` routines save and restore the terminal state, respectively. The `savetty()` routine saves the current state in a buffer; the `resetty()` routine restores the state to that stored in the buffer at the time of the last `savetty()` call.

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

scanw
wscanw
mvscanw
mvwscanw
vwscanw

The `scanw` set of routines is used to perform a formatted read from a window.

Syntax

```
#include <curses.h>

int scanw(char *fmt [,arg...]);
int wscanw(WINDOW *win, char *fmt [,arg...]);

int mvscanw(int y, int x, char *fmt[,arg...]);
int mvwscanw(WINDOW *win, int y, int x, char *fmt[,arg...])

#include <stdargs.h>

vwscanw(WINDOW *win, char *fmt, va_list arglist);
```

Parameters

fmt [,arg...] A `vwscanw()` format string, where *arg* is zero or more arguments used to satisfy the `scanf()` string.

fmt, arglist A `vscan()` format string, where *arglist* is a pointer to zero or more arguments used to satisfy the `scanf()` string.

win A pointer to the window in which the character is to be read.

x The *x* (column) coordinate of the position of the character to be read.

y The *y* (row) coordinate of the position of the character to be read.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

These routines are functionally equivalent to `scanf()`. Characters are read from the window using `wgetstr()`. When a newline is received, the line is processed by `scanw()`, which places the result in the appropriate *args*.

With `scanw()` and `wscanw()`, the characters are read from `stdscr` and *win*, respectively. The `mvscanw()` and `mvwscanw()` routines position the cursor in the window and then call `scanw()`. The `vwscanw()` routine reads from the window using the *stdargs* variable list.

Implementation Considerations

The `scanw()`, `wscanw()`, `mvscanw()`, and `mvwscanw()` routines are identical to XPG/3. The `vwscanw()` routine is a UNIX System V implementation.

See Also

`wgetstr()`

Portability

The `scanw()`, `wscanw()`, `mvscanw()`, and `mvwscanw()` routines conform to HP-UX, UNIX System V and XPG/3. The `vwscanw()` routine conforms to UNIX System V.

scr_dump
scr_restore

The `scr_dump` and `scr_restore` routines are used to write the screen contents to and from a file.

Syntax

```
#include <curses.h>

int scr_dump(char *filename);
int scr_restore(char *filename);
```

Parameters

filename	A pointer to the file in which screen contents are written.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `scr_dump()` routine writes the contents of the virtual screen, *curscr*, to *filename*. The `scr_restore()` routine writes the contents of *filename* (which must have been written with `scr_dump()`) to *curscr*. The next call to `doupdate()` restores the screen to the way it looks in *filename*.

Implementation Considerations

UNIX System V implementation

See Also

wrefresh()

Portability

UNIX System V

srcl
wscrl
scrol

The `srcl` set of routines is used to scroll a window.

Syntax

```
#include <curses.h>

int scrl (int n);
int wscrl (WINDOW *win, int n);

int scroll (WINDOW *win);
```

Parameters

win	A pointer to the window in which to scroll.
n	The number and direction of lines to scroll.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `scroll()` routine scrolls the window *win* up one line. The current cursor position is not changed.

The `scrl()` and `wscrl()` routines scroll the window `stdscr` or *win* up or down *n* lines, where *n* is a positive or negative integer, respectively.

The `scrollok()` routine must be enabled for these routines to work.

Note The `scroll()` and `scrl()` routines are macros.

Implementation Considerations

The `scroll()` routine is identical to XPG/3. The `wscrl()` and `scrl()` routines are UNIX System V implementations.

See Also

`scrollok()`, `waddch()`

Portability

The `scroll()` routine conforms to HP-UX, UNIX System V, and XPG/3. The `wscrl()` and `scrl()` routines conform to UNIX System V.

scrollok

The `scrollok` routine is used to enable scrolling of the screen.

Syntax

```
#include <curses.h>

int scrollok (WINDOW *win, bool bf);
```

Parameters

bf A Boolean expression.

win A pointer to the window in which to enable scrolling.

Return Values

`OK` Successful completion.

`ERR` An error occurred.

Description

The `scrollok()` routine controls what happens when the cursor advances outside the bottom boundary of a window or scrolling region. When enabled, if the cursor advances outside the bottom boundary of a window or scrolling region, a call to `wscrl()` scrolls up one line and updates the screen. If `scrollok()` is disabled, a call to the `wscrl()` routine leaves the cursor on the bottom of the line.

The terminal screen produces a scrolling effect if `idlok()` is also enabled.

Implementation Considerations

Identical to XPG/3

See Also

`idlok()`, `scroll()`, `waddch()`, `wscrl()`

Portability

HP-UX, UNIX System V, XPG/3

set_curterm

The `set_curterm` routine is used to set the *cur_term* variable (interface to `terminfo`).

Syntax

```
#include <curses.h>

int set_curterm (TERMINAL *nterm);
```

Parameters

nterm	The terminal type for which the variable is set.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `set_curterm()` routine is a low-level routine used only outside of `CURSES` when the program has to deal directly with the `terminfo` database to handle certain terminal capabilities. The use of appropriate `CURSES` routines is recommended in all other situations.

The `set_curterm()` routine sets the *cur_term* variable to *nterm*. The values from *nterm* as well as other state information for the terminal are used by `terminfo` functions such as `mvcur()`, `tigetflag()`, `tigetstr()`, and `tigetnum()`.

Implementation Considerations

UNIX System V implementation

See Also

`del_curterm()`

Portability

HP-UX, UNIX System V

set_term

The `set_term` routine is used to switch between terminals.

Syntax

```
#include <curses.h>

SCREEN *set_term (SCREEN *new);
```

Parameters

new	The new terminal to which to switch.

Return Values

On success, a pointer to the previous terminal is returned; otherwise, a null pointer is returned.

Description

The `set_term()` routine switches to the terminal specified by *new* and returns a screen reference to the previous terminal. Calls to subsequent `CURSES` routines affect the new terminal.

Implementation Considerations

Identical to XPG/3

Portability

HP-UX, UNIX System V, XPG/3

setscrreg

The `setscrreg` routine is used to set the scrolling region in a window.

Syntax

```
#include <curses.h>

int setscrreg (int top, int bot);
int wssetscrreg (WINDOW *win, int top, int bot);
```

Parameters

bot	The bottom line of the scrolling region (top of the window is line 0).
top	The top line of the scrolling region (top of the window is line 0).
win	A pointer to the window in which to set up the scroll window.

Return Values

None

Description

The `setscrreg()` and `wsetscrreg()` routines set up scrolling regions in the windows `stdscr` and *win*, respectively. The dimensions of the scrolling region are defined by the *top* and *bot* parameters. If `scrollok()` is active, any attempt to move the cursor beyond the bottom margin of the scrolling region scrolls the text in the scrolling region up one line.

The terminal screen produces a scrolling effect if `idlok()` is also enabled.

Implementation Considerations

Identical to XPG/3

See Also

`idlok()`, `scroll()`, `scrollok()`, `waddch()`, `wscrl()`

Portability

HP-UX, UNIX System V, XPG/3

setupterm
setterm

The `setupterm` and `setterm` routines are used to define a set of terminal-dependent variables (`terminfo` interface).

Syntax

```
#include <curses.h>

int setupterm (char *term, int fildes, int *errret);
int setterm (char *term);
```

Parameters

term	The terminal type for which variables are set.
fildes	A file descriptor initialized for output.
errret	A pointer to an integer in which the status value is stored.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

Within `CURSES`, `setupterm()` is automatically called by `initscr()` and `newterm()`. This routine can be also be used outside of `CURSES` when the program has to deal directly with the `TERMINFO` database to handle certain terminal capabilities. The use of appropriate `CURSES` routines is recommended in all other situations.

The `setupterm()` routine defines terminal-dependent variables for the `terminfo` layer of `CURSES`. The `setupterm()` routine initializes the `terminfo` variables lines and columns such that if `use_env(FALSE)` has been called, the `TERMINFO` values are used regardless of the environmental variables `LINES` and `COLUMNS` or the program's window dimensions. When `use_env(TRUE)` has been called, which is the default, the environment variables `LINES` and `COLUMNS` are used, if they exist. If the environment variables do not exist and the program is running in a window, the current window size is used.

The *term* parameter of `setupterm()` specifies the terminal; if null, terminal type is taken from the `TERM` environment variable. All output is sent to *fildes* which is initialized for output. If *errret* is not null, `OK` or `ERR` is returned and a status value is stored in the integer pointed to by *errret*. The status values that may be returned are shown in Table 4-7.

Table 4-7. Status Values

Value	Description
1	Normal
0	Terminal could not be found
-1	`TERMINFO` database could not be found

If *erret* is null, an error message is printed, and `setupterm()` exits. The `setterm()` routine is an older version of `setupterm()`. It is included for compatibility with previous versions of `CURSES`. New programs should use `setupterm()`.

Implementation Considerations

HP-UX and UNIX System V implementations.

See Also

`use_env()`

Portability

HP-UX, UNIX System V

start_color

The `start_color` routine is used to initialize the use of color.

Note The `start_color` routine is not implemented at this time.

Syntax

```
#include <curses.h>

int start_color();
```

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The `start_color()` routine initializes the use of color. It must be used if color is to be used in the program. It must be called before any other color routines, ideally right after `initscr()`. Eight basic colors are initialized (black, red, green, yellow, blue, magenta, cyan, and white) and two global variables (`COLORS` and `COLOR_PAIRS`). The `COLORS` variable specifies the number of colors that the terminal supports, and the `COLOR_PAIRS` variable specifies the number of color pairs. Colors are always in pairs consisting of a foreground color (for characters) and a background color (for the the rest of the character cell). The `start_color()` routine also restores the values of the colors to those that the terminal had on startup.

Implementation Considerations

UNIX System V implementation

See Also

can_change_color(), color_content(), has_color(), init_color(), init_pair(), pair_content()

Portability

UNIX System V

subwin

The `subwin` routine is used to create a subwindow relative to the physical screen.

Syntax

```
#include <curses.h>

WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int begin_y,
int begin_x);
```

Parameters

orig	The parent window of the subwindow.
nlines	The number of lines in the subwindow.
ncols	The number of columns in the subwindow
begin_y	The *y* (row) coordinate of the upper-left corner of the window.
begin_x	The *x* (column) coordinate of the upper-left corner of the window.

Return Values

On success, a pointer to the new window structure is returned; otherwise, a null pointer is returned.

Description

The `subwin()` routine creates a subwindow within window *orig*, with the specified number of lines and columns, and upper left corner positioned at *begin_x*, *begin_y* (relative to the physical screen, not to window *orig*). A pointer to the new window structure is returned.

The original window and subwindow share character storage of the overlapping area. (Each window maintains its own pointers, cursor location, and other items.) This means that characters and attributes are identical in overlapping areas regardless of which window characters are written to.

When using subwindows, it is often necessary to call `touchwin()` before `wrefresh()` to maintain proper screen contents.

Note The `derwin()` routine creates a subwindow in exactly the same way, but allows you to specify coordinates relative to window *orig*.

Implementation Considerations

Identical to XPG/3

See Also

`newwin()`, `touchwin()`, `derwin()`

Portability

HP-UX, UNIX System V, XPG/3

termattrs

The `termattrs` routine is used to return the video attributes supported by the terminal.

Syntax

```
#include <curses.h>

chtype termattrs();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `termattrs()` routine returns a logical `OR` of all video attributes available on a terminal.

Implementation Considerations

UNIX System V implementation

Portability

UNIX System V

termname

The `termname` routine is used to obtain the return value of the environmental variable `TERM`.

Syntax

```
#include <curses.h>

char *termname();
```

Return Values

On success, a pointer to the value of the environmental variable is returned; otherwise, a null pointer is returned.

Description

The `termname()` routine returns a pointer to the value of the environmental variable `TERM` (truncated to 14 characters).

Implementation Considerations

UNIX System V implementation

Portability

UNIX System V

tgetent

The `tgetent` routine is used to look up the `termcap` name (interface to `termcap` library).

Syntax

```
#include <curses.h>

int tgetent (char *bp, char *name);
```

Parameters

bp	A pointer to a buffer 1024 bytes long.
name	The `termcap` entry to look up.

Return Values

-1	Cannot open `termcap` file.
0	No entry in `termcap`.
1	Successful completion.

Description

The `tgetent()` routine looks up the `termcap` entry for the terminal name.

The `tgetent()` routine is included for compatibility with programs that use the `termcap` library. New programs should use `terminfo` functions.

Note The parameter *bp* should designate an area 1024 bytes long that is retained for all calls to `tgetnum()`, `tgetflag()`, `tgetstr()`, and `tgoto()`.

Implementation Considerations

UNIX System V implementation

See Also

`setupterm()`

Portability

HP-UX, UNIX System V

tgetflag

The `tgetflag` routine is used to get the Boolean entry for `termcap` capability (interface to `termcap` library).

Syntax

```
#include <curses.h>

int tgetflag (char id[2]);
```

Parameters

cap	The capability for which to get the Boolean entry.

Return Values

`TRUE`	Successful completion.
`FALSE`	An error occurred.

Description

The `tgetflag()` routine returns the Boolean value of the `termcap` *cap*.

The `tgetflag()` routine is included for compatibility purposes with programs that use the `termcap` library. New programs should use the `terminfo` routines.

Implementation Considerations

UNIX System V implementation

See Also

`tigetflag()`, `tputs()`

Portability

HP-UX, UNIX System V

tgetnum

The `tgetnum` routine is used to get the numeric entry for `termcap` capability (interface to `termcap` library).

Syntax

```
#include <curses.h>

int tgetnum (char id[2]);
```

Parameters

cap	The `termcap` capability for which to get the numeric entry.

Return Values

Returns the value of the numeric `termcap` entry, or returns -1 if the entry is not given for the terminal.

Description

The `tgetnum()` routine looks up the numeric entry for *cap*.

The `tgetnum()` routine is included for compatibility purposes with programs that use the `termcap` library. New programs should use the `terminfo` routines.

Implementation Considerations

UNIX System V implementation

See Also

`tigetnum()`, `tputs()`

Portability

HP-UX, UNIX System V

tgetstr

The `tgetstr` routine is used to get the string entry for `termcap` capability (interface to `termcap` library).

Syntax

```
#include <curses.h>

char *tgetstr (char cap[2], char **area);
```

Parameters

cap	The `termcap` capability for which to get the string entry.
area	A pointer to the area where the decoded string is stored.

Return Values

On success, a pointer to the string is returned; otherwise, a null pointer is returned.

Description

The `tgetstr()` routine looks up the string entry for the `termcap` *cap*, placing the decoded string at *area* and advancing the area pointers. The `tputs()` routine should be used to output the string.

The `tgetstr()` routine is included for compatibility purposes with programs that use the `termcap` library. New programs should use the `terminfo` routines.

Implementation Considerations

UNIX System V implementations

See Also

`tigetstr()`, `tputs()`, `tparm()`

Portability

HP-UX, UNIX System V

tgoto

The `tgoto` routine is used to decode the cursor motion values (interface to `termcap` library).

Syntax

```
#include <curses.h>

char *tgoto (char *cap, int col, int row);
```

Parameters

cap	The pointer to the `termcap` capability for cursor motion.
col	The column placement of the new cursor.
row	The row placement of the new cursor.

Return Values

On success, a pointer to the decoded cursor addressing string is returned; otherwise, a null pointer is returned.

Description

The `tgoto()` routine decodes cursor values returned from `tgetstr()`. A pointer to a cursor addressing string is returned that, when sent to the terminal with `tputs()`, moves the cursor to the new location.

The `tgoto()` routine is included for compatibility purposes with programs that use the `termcap` library. New programs should use `terminfo` routines.

Implementation Considerations

UNIX System V implementation

See Also

`mvcur()`

Portability

HP-UX, UNIX System V

tigetflag
tigetnum
tigetstr

The `tigetflag`, `tigetnum`, and `tigetstr` routines are used to return the value of `terminfo` capability (interface to `terminfo`).

Syntax

```
#include <curses.h>

int tigetflag (char *capname);
int tigetnum (char *capname);
char *tigetstr (char *capname);
```

Parameters

capname	The name of the `terminfo` capability for which the value is required.

Return Values

The `tigetflag()` routine returns -1 if *capname* is not a Boolean capability. The `tigetnum()` routine returns -1 if *capname* is not a numeric capability. The `tigetstr()` routine returns `(char *)-1` if *capname* is not a string capability.

Description

The `tigetflag()`, `tigetnum()`, and `tigetstr()` routines return values for `terminfo` capabilities passed to them.

The following null-terminated arrays contain the *capnames*, the *termcap* codes and full C names for each of the `terminfo` variables.

```
char *boolnames, *boolcodes, *boolfnames
char *numnames, *numcodes, *numfnames
char *strnames, *strcodes, *strfnames
```

Implementation Considerations

UNIX System V implementation

See Also

`terminfo`

Portability

UNIX System V

timeout
wtimeout

The `timeout` and `wtimeout` routines are used to set a timed blocking or nonblocking read for a window.

Syntax

```
#include <curses.h>

int timeout(int delay);
int wtimeout(WINDOW win, int delay);
```

Parameters

delay	The number of milliseconds to block or wait for input.
win	A pointer to the window in which to set the timed blocking.

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The `timeout()` and `wtimeout()` routines set the length of time `getch()` waits for input for windows `stdscr` and *win*, respectively. These routines are similar to `nodelay` except the time to block or wait for input can be specified.

A negative delay causes the program to wait indefinitely for input; a delay of 0 returns ERR if no input is ready; and a positive delay blocks until input arrives or the time specified expires (in which case, ERR is returned).

Note The `timeout()` routine is a macro.

Implementation Considerations

UNIX System V implementation

See Also

`wgetch()`, `nodelay()`

Portability

UNIX System V

touchwin
touchline
untouchwin
wtouchln
is_linetouched
is_wintouched

The `touchwin` set of routines is used to control the refresh of the window.

Syntax

```
#include <curses.h>

int touchwin(WINDOW *win);
int touchline(WINDOW *win, int start, int count);
int untouchwin(WINDOW *win);
int wtouchln(WINDOW *win, int y, int n, int changed);

int is_linetouched(WINDOW *win, int line);
int is_wintouchwin(WINDOW *win);
```

Parameters

count The number of lines in the window to mark as changed.

changed A flag indicating whether to make lines look changed (`TRUE`) or not changed (`FALSE`).

line The line to be checked for change since refresh.

n The number of lines in the window to mark as changed.

win A pointer to the window in which the refresh is to be controlled or monitored.

start The starting line number of the portion of the window to make appear changed.

y The starting line number of the portion of the window to make appear changed or not changed.

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The `touchwin()` routine marks the entire window as dirty. This makes it appear to CURSES as if the whole window has been changed, thus causing the entire window to be rewritten with the next call to `wrefresh()`. This is sometimes necessary when using overlapping windows; the change to one window is not reflected in the other and, hence is not recorded.

The `touchline()` routine marks as dirty a portion of the window starting at line *start* and continuing for *count* lines, instead of the entire window. Consequently, that portion of the window is updated with the next call to `wrefresh()`.

The `untouchwin()` routine marks all lines in the window as unchanged since the last refresh, ensuring that it is not updated.

The `wtouchln()` routines marks *n* lines starting at line *y* as either changed (TRUE) or unchanged (FALSE) since the last refresh.

To find out which lines or windows have been changed since the last refresh, use the `is_linetouched()` and `is_wintouched()` commands, respectively. These return TRUE if the specified lines or window have been changed since the last call to `wrefresh()` or FALSE if no changes have been made.

Note All routines except `wtouchln()` and `is_wintouched()` are macros.

Implementation Considerations

The `touchwin()` routine is identical to XPG/3. The `touchline()`, `untouchwin()`, `wtouchln()`, `is_linetouched()`, and `is_wintouched()` routines are UNIX System V implementations.

See Also

`wrefresh()`

Portability

The `touchwin()` routine conforms to HP-UX, UNIX System V, and XPG/3. The `touchline()`, `untouchwin()`, `wtouchln()`, `is_linetouched()`, and `is_wintouched()` routines conform to UNIX System V.

tparm

The `tparm` routine is used to instantiate a parameterized string (interface to `terminfo`).

Syntax

```
#include <curses.h>

char *tparm (char *str, long int p1, long int p2, long int p3,
   long int p4, long int p5, long int p6, int p7,
   long int p8, long int p9);
```

Parameters

p1...p9 The parameters to be instantiated.

str A pointer to the string to be instantiated.

Return Values

On success, a pointer to parameterized string is returned; otherwise, a null pointer is returned.

Description

The `tparm()` routine is a low-level routine used outside of `CURSES` to deal directly with the `terminfo` database. The use of appropriate `CURSES` routines is recommended for most situations.

The `tparm()` routine instantiates a parameterized string using up to nine arguments. The string is suitable for output processing by `tputs()`.

Implementation Considerations

UNIX System V implementation

See Also

`tputs()`

Portability

HP-UX, UNIX System V

tputs
putp

The `tputs` and `putp` routines are used to apply padding information and output string (interface to `terminfo`).

Syntax

```
#include <curses.h>

int tputs (char *str, int affcnt, int (*putc) (int));
int putp (char *str);
```

Parameters

str	A pointer to a `terminfo` variable or return value from `tparm()` or `tigetstr()`.
affcnt	The number of lines affected, or 1 if not relevant.
putc	The output function.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `tputs()` and `putp()` routines are low-level routines used outside of `CURSES` to deal directly with the `terminfo` database. The use of appropriate `CURSES` routines is recommended for most situations.

The `tputs()` routine adds padding information and then outputs *str*. The value for *str* must be the result value from `tparm()`, or `tigetstr()`, or a `terminfo` string variable.

The `tputs()` routine replaces the padding specification (if one exists) with enough characters to produce the specified delay. Characters are output one at a time to `putc`, a routine similar to `putchar()`.

tputs

The `putp()` routine calls `tputs()` as follows:

```
tputs(str, 1, _putchar)
```

Note The output of `putp()` goes to `stdout`, not to the file descriptor, *fildes*, specified in `setupterm()`.

Implementation Considerations

UNIX System V implementation

See Also

`tigetstr()`, `tparm()`

Portability

HP-UX, UNIX System V

traceon
traceoff

The `traceon` and `traceoff` routines are used to enable or disable tracing.

Syntax

```
#include <curses.h>

traceoff();
traceon();
```

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `traceon()` and `traceoff()` routines turn on or turn off tracing, respectively. Since the volume of output produced by these routines can be very large, limit the area traced at a given time.

The output generated by these routines is stored in a file called `trace.out`. The information provided by the `trace.out` file is the name of the function and its parameters when entered and the function name and its return value upon exit.

Implementation Considerations

Nonstandard

Portability

HP-UX

typeahead

The `typeahead` routine is used to check for type-ahead characters.

Syntax

```
#include <curses.h>

int typeahead (int fd);
```

Parameters

fd	The file descriptor that is used to check for type-ahead characters.

Return Values

`OK`	Successful completion.
`ERR`	An error occurred.

Description

The `typeahead()` routine specifies the file descriptor (*fd*) to use to check for type-ahead characters (characters typed by the user but not yet processed by `CURSES`).

`CURSES` checks for type-ahead characters periodically while updating the screen. If characters are found, the current update is postponed until the next `wrefresh()` or `doupdate()`. This speeds up response to commands that have been typed ahead. Normally, the input `FILE` passed to `newterm()`, or `stdin` in the case of `initscr()`, is used for type-ahead checking.

If *fd* is -1, no type-ahead checking is done.

Implementation Considerations

Identical to XPG/3

See Also

`wgetch()`, `wrefresh()`

Portability

HP-UX, UNIX System V, XPG/3

unctrl

The `unctrl` routine is used to convert a character to printable form.

Syntax

```
#include <curses.h>
#include <unctrl.h>

char *unctrl(chtype c);
```

Return Values

Returns a string.

Description

The `uncntl()` routine converts the character code *c* into a printable form (if unprintable). Control characters are displayed in the ^*x* notation where ^ identifies the control key, and *x* represents an alphanumeric character that is pressed while the control key is held down.

Implementation Considerations

Identical to XPG/3

See Also

`waddch()`, `waddstr()`

Portability

HP-UX, UNIX System V, XPG/3

use_env

The use_env routine is used to set values of lines and columns.

Syntax

```
#include <curses.h>

int use_env(char bool);
```

Parameters

bool	A Boolean expression.

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The use_env() routine takes the values for lines and columns from the terminfo database (if *bool* is FALSE), or from environmental variables (if *bool* is TRUE). If no environmental variables have been set, the window size is used. This routine must be set before initscr(), newterm(), or setupterm() is called. The default action is TRUE.

Implementation Considerations

UNIX System V implementation

See Also

initscr(), newterm(), setupterm()

Portability

UNIX System V

vidputs
vidattr

The vidputs and vidattr routines are used to display a string with video attributes (interface to terminfo).

Syntax

```
#include <curses.h>

int vidputs (chtype attrs, int (*putc) (int));
int vidattr (chtype attrs);
```

Parameters

attrs	The attributes of the foreground window.
putc	The output function.

Return Values

OK	Successful completion.
ERR	An error occurred.

Description

The vidputs() and vidattr() routines are low-level routines used outside of CURSES to deal directly with the terminfo database. The use of appropriate CURSES routines is recommended for most situations.

The vidputs() routine enables and disables attributes for the current terminal. The attributes shown in Table 4-8 are defined in curses.h and can be combined with the bitwise OR operator.

Table 4-8. Constant Values for Highlighting Attributes

Constant	Description
`A_ALTCHARSET`	Alternate character set
`A_ATTRIBUTES`	Attribute mask
`A_BLINK`	Blinking
`A_BOLD`	Bold
`A_CHARTEXT`	Character mask
`A_COLOR`	Color mask
`A_DIM`	Dim
`A_INVIS`	Invisible
`A_NORMAL`	Disable attributes
`A_PROTECT`	No display
`A_REVERSE`	Reverse video
`A_STANDOUT`	Highlights specific to terminal
`A_UNDERLINE`	Underline
`COLOR_PAIR`(n)	Color-pair number n
`PAIR_NUMBER`(a)	Pair number for `COLOR_PAIR`(n)

The characters are passed one at a time to the `putc` routine, a routine similar to `putchar()`.

The `vidattr()` routine is similar to `vidputs()` except characters are sent to `stdout` instead of a user-supplied output function.

Implementation Considerations

UNIX System V implementation

Portability

HP-UX, UNIX System V

Index

D

E

F

S

W

FINAL TRIM SIZE : 7.0 in x 8.5 in