NAME

Term::ReadKey - A perl module for simple terminal control

SYNOPSIS

```
use Term::ReadKey;
ReadMode 4; # Turn off controls keys
while (not defined ($key = ReadKey(-1))) {
# No key yet
}
print "Get key $key\n";
ReadMode 0; # Reset tty mode before exiting
```

DESCRIPTION

Term::ReadKey is a compiled perl module dedicated to providing simple control over terminal driver modes (cbreak, raw, cooked, etc.,) support for non-blocking reads, if the architecture allows, and some generalized handy functions for working with terminals. One of the main goals is to have the functions as portable as possible, so you can just plug in ‘‘use Term::ReadKey’’ on any architecture and have a good likelihood of it working.

Version 2.30.01: Added handling of arrows, page up/down, home/end, insert/delete keys under Win32. These keys emit xterm-compatible sequences. Works with Term::ReadLine::Perl.

ReadMode MODE [, Filehandle]

Takes an integer argument or a string synonym (case insensitive), which can currently be one of the following values:

```
INT SYNONYM DESCRIPTION

0 'restore' Restore original settings.

1 'normal' Change to what is commonly the default mode,
echo on, buffered, signals enabled, Xon/Xoff
possibly enabled, and 8-bit mode possibly disabled.

2 'noecho' Same as 1, just with echo off. Nice for
reading passwords.

3 'cbreak' Echo off, unbuffered, signals enabled, Xon/Xoff
possibly enabled, and 8-bit mode possibly enabled.

4 'raw' Echo off, unbuffered, signals disabled, Xon/Xoff
disabled, and 8-bit mode possibly disabled.

5 'ultra-raw' Echo off, unbuffered, signals disabled, Xon/Xoff
disabled, 8-bit mode enabled if parity permits,
and CR to CR/LF translation turned off.
```

These functions are automatically applied to the STDIN handle if no other handle is supplied. Modes 0 and 5 have some special properties worth mentioning: not only will mode 0 restore original settings, but it cause the next ReadMode call to save a new set of default settings. Mode 5 is similar to mode 4, except no CR/LF translation is performed, and if possible, parity will be disabled (only if not being used by the terminal, however. It is no different from mode 4 under Windows.)

If you just need to read a key at a time, then modes 3 or 4 are probably sufficient. Mode 4 is a tad more flexible, but needs a bit more work to control. If you use ReadMode 3, then you should install a SIGINT or END handler to reset the terminal (via ReadMode 0) if the user aborts the program via `^C`. (For any mode, an END handler consisting of ‘‘ReadMode 0’’ is actually a good idea.)

If you are executing another program that may be changing the terminal mode, you will either want to

say

```
ReadMode 1; # same as ReadMode 'normal'
system('someprogram');
ReadMode 1;
```

which resets the settings after the program has run, or:

```
$somemode=1;
ReadMode 0; # same as ReadMode 'restore'
system('someprogram');
ReadMode 1;
```

which records any changes the program may have made, before resetting the mode.

ReadKey MODE [, Filehandle]

Takes an integer argument, which can currently be one of the following values:

```
0 Perform a normal read using getc
-1 Perform a non-blocked read
>0 Perform a timed read
```

If the filehandle is not supplied, it will default to STDIN. If there is nothing waiting in the buffer during a non-blocked read, then undef will be returned. In most situations, you will probably want to use `ReadKey -1`.

NOTE that if the OS does not provide any known mechanism for non-blocking reads, then a `ReadKey -1` can die with a fatal error. This will hopefully not be common.

If MODE is greater then zero, then ReadKey will use it as a timeout value in seconds (fractional seconds are allowed), and won't return `undef` until that time expires.

NOTE, again, that some OS's may not support this timeout behaviour.

If MODE is less then zero, then this is treated as a timeout of zero, and thus will return immediately if no character is waiting. A MODE of zero, however, will act like a normal getc.

NOTE, there are currently some limitations with this call under Windows. It may be possible that non-blocking reads will fail when reading repeating keys from more then one console.

ReadLine MODE [, Filehandle]

Takes an integer argument, which can currently be one of the following values:

```
0 Perform a normal read using scalar(<FileHandle>)
-1 Perform a non-blocked read
>0 Perform a timed read
```

If there is nothing waiting in the buffer during a non-blocked read, then undef will be returned.

NOTE, that if the OS does not provide any known mechanism for non-blocking reads, then a `ReadLine 1`can die with a f atal error. This will hopefully not be common.

NOTE that a non-blocking test is only performed for the first character in the line, not the entire line. This call will probably **not** do what you assume, especially with `ReadMode` MODE values higher then 1. For example, pressing Space and then Backspace would appear to leave you where you started, but any timeouts would now be suspended.

This call is currently not available under Windows.

GetTerminalSize [Filehandle]

Returns either an empty array if this operation is unsupported, or a four element array containing: the width of the terminal in characters, the height of the terminal in character, the width in pixels, and the height in pixels. (The pixel size will only be valid in some environments.)

NOTE, under Windows, this function must be called with an **output** filehandle, such as `STDOUT`, or a

handle opened to `CONOUT$`.

SetTerminalSize WIDTH,HEIGHT,XPIX,YPIX [, Filehandle]
Return -1 on failure, 0 otherwise.

NOTE that this terminal size is only for **informative** value, and changing the size via this mechanism will **not** change the size of the screen. For example, XTerm uses a call like this when it resizes the screen. If any of the new measurements vary from the old, the OS will probably send a SIGWINCH signal to anything reading that tty or pty.

This call does not work under Windows.

GetSpeed [, Filehandle]
Returns either an empty array if the operation is unsupported, or a two value array containing the terminal in and out speeds, in **decimal**. E.g, an in speed of 9600 baud and an out speed of 4800 baud would be returned as (9600,4800). Note that currently the in and out speeds will always be identical in some OS's.

No speeds are reported under Windows.

GetControlChars [, Filehandle]
Returns an array containing key/value pairs suitable for a hash. The pairs consist of a key, the name of the control character/signal, and the value of that character, as a single character.

This call does nothing under Windows.

Each key will be an entry from the following list:

```
DISCARD
DSUSPEND
EOF
EOL
EOL2
ERASE
ERASEWORD
INTERRUPT
KILL
MIN
QUIT
QUOTENEXT
REPRINT
START
STATUS
STOP
SUSPEND
SWITCH
TIME
```

Thus, the following will always return the current interrupt character, regardless of platform.

```
%keys = GetControlChars;
$int = $keys{INTERRUPT};
```

SetControlChars [, Filehandle]
Takes an array containing key/value pairs, as a hash will produce. The pairs should consist of a key that is the name of a legal control character/signal, and the value should be either a single character, or a number in the range 0-255. SetControlChars will die with a runtime error if an invalid character name is passed or there is an error changing the settings. The list of valid names is easily available via

```
%cchars = GetControlChars();
@cnames = keys %cchars;
```

This call does nothing under Windows.

AUTHOR

Kenneth Albanowski <kjahds@kjahds.com>

Currently maintained by Jonathan Stowe <jns@gellyfish.co.uk>

SUPPORT

The code is maintained at

https://github.com/jonathanstowe/TermReadKey

Please feel free to fork and suggest patches.

LICENSE

Prior to the 2.31 release the license statement was:

```
Copyright (C) 1994-1999 Kenneth Albanowski.
2001-2005 Jonathan Stowe and others

Unlimited distribution and/or modification is allowed as long as this
copyright notice remains intact.
```

And was only stated in the README file.

Because I believe the original author's intent was to be more open than the other commonly used licenses I would like to leave that in place. However if you or your lawyers require something with some more words you can optionally choose to license this under the standard Perl license:

```
This module is free software; you can redistribute it and/or modify it
under the terms of the Artistic License. For details, see the full
text of the license in the file "Artistic" that should have been provided
with the version of perl you are using.

This program is distributed in the hope that it will be useful, but
without any warranty; without even the implied warranty of merchantability
or fitness for a particular purpose.
```