NAME

pcmapkeys – set keyboard extended map and scancode translation for the PC console in text mode

SYNOPSIS

pcmapkeys [**–f** *mapfile* | **–n** | **–g** | **–d** | **–e**]

AVAILABILITY

x86

SUNWcsu

DESCRIPTION

pcmapkeys is a utility that permits a user to activate character mapping on input and output and keyboard extended mapping on the PC console in text mode. The keyboard extended mapping consists of the support for the deadkey and compose key sequences.

Consistent Keyboard-Display Mapping

The original UNIX operating system was written to support the ASCII codeset. ASCII is one of many standards to represent a number of characters internally as certain numbers. Typical for ASCII is that it supports 128 different characters, each represented by a single byte of which the 8^{th} bit is not used. Many UNIX system applications, including the shell, took advantage of this. Starting with UNIX System V Release 3.1, most of these applications have been modified to properly support characters represented as a byte with the 8^{th} bit set as well. This means that now 256 characters can be supported at the same time. However, a consistent coding convention needs to be applied. In the IBM PC world, an 8-bit coding referred to as IBM extended ASCII has been used for several years; MS-DOS users are quite familiar with that. In heterogeneous UNIX system environments, a different codeset, called ISO 8859, has been promoted. In both codesets, characters found in the ASCII codeset are represented in the same way. The other 128 characters are encoded differently, however, and some characters found in one codeset will be missing in the other. The Solaris for x86 system supports both codesets; actually, it supports any 8-bit one byte codeset.

To be able to use characters from the French, German, Finnish, and other alphabets, there are systems available on the market that generate 7-bit codes but display the above-mentioned characters on the screen instead of the ones found on a U.S. console. On the keyboard there are an equal number of keys, but there are different characters on the key caps. Others may support 256 different characters at a time but use their own proprietary codesets.

For example, if you are using the Solaris for x86 system with a console and a French keyboard and you do not use **pcmapkeys** to map the French keyboard tables, then if you edit a file and use the French character *é* in text, the actual code generated is ASCII 123, which is the code normally used for the left curly brace. If you look at the edited file on the console, the letter will actually appear to be a curly brace. Using **pcmapkeys** to map in the French keyboard allows consistent input and output mappings.

Input mapping

On input, any byte can be mapped to any byte. Using the example above, you could map 123 to 130, the code used for *é* in the IBM extended ASCII codeset.

Output mapping

On output, any byte can be mapped to either a byte or a string. In the above example, 130 would be mapped back to 123 to properly display the character on the screen. If the connected device is a printer that does not support the *é* character, it could be mapped to the string '**e BACKSPACE**'.

Deadkeys

On typewriters, keys can be found that behave slightly differently than all the others, because when you press them, the printing wheel of the typewriter does not move. Ctrl (^) and the grave accent (`) are such characters. When ` is followed by an *e*, the letter *è* is generated. This is called a deadkey or a non-spacing character. Solaris for x86 supports the use of deadkeys. Typically, the ^ character, the ` character, and the umlaut character are used as deadkeys.

Compose sequences

Characters can also be generated using a compose sequence. A dedicated character called the ''compose character'' followed by two other keystrokes will generate a single character. As an example, COMPOSE followed by the plus and the minus sign could generate the plus/minus sign (±). Compose sequences can also be used as an alternative for deadkeys, e.g., ''COMPOSE ^ e'' instead of ''^ e.''

Numeric compose sequences

Compose sequence characters that are not present on the keyboard and cannot be intuitively composed by some key sequence, for example, graphics characters, can be generated by pressing the compose key followed by three digits.

Toggle key

An optional toggle key can be defined to temporarily disable the current mapping from within an application. This can be useful when, for example, a German programmer wants easy access to the curly braces and the brackets. Use of the toggle key is analogous to the use of the −**d** and −**e** command line options.

Scancode Mapping

The keyboards of the console and some other peripherals such as SunRiver workstations behave differently than those of regular terminals. They generate what are called *scancodes* and you will also find a number of keys on these keyboards, such as the Alt key, that are not found on regular terminals. Scancodes generated by PC keyboards typically represent the location of the key on the keyboard. The keyboard driver has to properly translate these scancodes. The different national variants of a PC keyboard not only have non-English characters printed on some of the keycaps, but the order of some of the keys is different as well. Without changing the scancode translation, a French user would type **A** and see **Q** on his screen. Several status keys can influence the translated code as well. The keyboard driver, and thus the **pcmapkeys** program, makes a distinction between two sets of key combinations that can be translated.

Function keys

Up to 60 key combinations are recognized as function keys. The first 12 are the 12 function keys of a 101-key PC-keyboard (the first 10 on an 84-key keyboard).

If you do not know whether you have an 84- or 101-key keyboard, you can use the following scheme to determine which type you have:

If your keyboard has arrow keys that are separate from the ones on the numeric keypad, then you have a 101-key keyboard.

If the arrow keys on your keyboard are located on the numeric keypad only, then you have an 84-key keyboard.

F13 to F24 are the same keys used in combination with Shift, F25 to F36 when used with Ctrl, and F37 to F48 when used with Ctrl and Shift together. F49 to F60 are the keys on the numeric keypad, in the following order:

7
8
9
–
4
5
6
+
1
2
3
INS

Each of these function keys can be given a string as a value. The total length of all strings should not exceed 512 characters.

Regular keys
Scancodes generated by all keys on the PC keyboard can be translated in a different way as well. For each key, a different translation can be specified for each of the following four cases:

1. The key is pressed.
2. The key and the Shift key are pressed simultaneously.
3. The key and the Alt key are pressed simultaneously.
4. The key, the Shift, and the Alt keys are pressed simultaneously.

For each of these cases, the scancode can be translated into one of the following:

a single byte
a single byte preceded by ESC N
a single byte preceded by ESC O
a single byte preceded by ESC [

Internally, special bits are set to indicate that an escape sequence needs to be generated. Other bits are used to indicate whether the translated code should be influenced by some special keys.

Num Lock
If the Num Lock bit is set, the regular and Shift values are swapped, as are the Alt and Shift Alt values, whenever the Num Lock LED is on. By default, only the keys on the numeric keypad have this bit set. That is why these keys generate 7,

8, 9, etc. when the Num Lock LED is on, which is the same value that would be produced if Shift were used with these keys.

Caps Lock
: This has the same effect as the Num Lock key. By default, this bit is set for all letters and not set for punctuation signs.

Ctrl
: When a key is translated into a single byte (no escape sequence) and this bit is set, the corresponding control character will be generated when the Ctrl key is pressed simultaneously. This is equally valid for the Shift, Alt, and Shift Alt combination. When this bit is not used, the Ctrl key combination will not generate anything.

mapfile

This section describes the layout of a *mapfile* that is read by the **pcmapkeys** program.

A *mapfile* is a text file that consists of several **sections**. A sharp sign (**#**) can be used to include comments. Everything following the **#** until the end of the line will be ignored by the **pcmapkeys** program. Inside a line, C-style comments can be used as well. The beginning of each section is indicated by a *keyword*. Spaces and tabs are silently ignored and can be used at all times to improve readability. All but one section, the one that defines the *compose character*, can be left out. The order in which the different sections should appear is predefined. Here is the list of keywords in the order they should appear:

input:
toggle:
dead:
compose:
output:
scancodes:

Characters can be described in several different ways. ASCII characters can be described by putting them between single quotes. For example:

’a’ ’{’

Between single quotes, control characters can be listed by using a circumflex sign before the character that needs to be quoted. For example:

’^x’

When a backslash (\) is used, what follows will be interpreted as a decimal, octal (leading zero), or hexadecimal (leading x or X) representation of the character, although in this case the use of single quotes is not mandatory. For example:

’\x88’

is the same as:

0x88 (zero needed when not quoted)

and:

’\007’

is the same as:

```
007
```

When strings are needed, a list of character representations should be used. Quoted strings will be supported in the future.

The following paragraphs describe what goes in each section.

Input section
The input section describes which input characters should be mapped into a single byte. A very small sample input section could be:

```
input:
'A'   'B'      # map A into B on input
'#'   0x9c     # map sharp sign into pound sign
```

Toggle section
The toggle section is a one-line section that defines which key is to toggle between mapping and no mapping. For example:

```
toggle:
'^y'           # ctrl y is the toggle key
```

Deadkey section
The deadkey section defines which keys should be treated as deadkeys. A **dead:** keyword followed by the specification of the character appears in this section for each deadkey. The subsequent lines describe what key should be generated for each key following the deadkey. A deadkey followed by a key not described in this part of the *mapfile* will not generate any key and a beep tone will be produced on the terminal. For example:

```
dead:  '^'    # circumflex is a deadkey
' '    '^'    # circumflex followed by space generates circumflex
'e'    0x88   # circumflex followed by e generates e circumflex
dead:  '"'    # double quote used as a deadkey
' '    '"'    # double quote space generates double quote
'a'    0x84   # double quote a generates an umlaut
```

Compose section
The first line of this section describes what the compose character is. That line should always be present in the *mapfile*. Subsequent lines consist of three character representations indicating each time that the third character needs to be generated on input when the compose character is followed by the first two. Compose sequences with the same first character should be grouped together. For example:

```
compose: '^x'
'"'  'e'  0x89   # e with umlaut is generated when typing ^x " e
'"'  'a'  0x84   # a with umlaut
'e'  '"'  0x89   # e with umlaut is generated when typing ^x e "
'a'  '"'  0x84   # a with umlaut
```

The following example would give the wrong result. All lines starting with the same character specification should be grouped together.

```
compose: '^x'
'"'   'e'    0x89    # e with umlaut is generated when typing ^x " e
'e'   '"'    0x89    # e with umlaut is generated when typing ^x e "
'"'   'a'    0x84    # a with umlaut
'a'   '"'    0x84    # a with umlaut
```

Output section
This section describes the mapping on output, either single byte to single byte, or single byte to string. A string is specified as a series of character specifications. For example:

```
output:
0x82 '{'                # map e with accent to { to display e with accent
'^u' '(''K''I''L''L'')'  # print (KILL) when kill character is used
```

Scancodes section
This section will only have an effect when your terminal is a scancode device. No error message will be produced if this section is in your *mapfile* when not needed, because the **pcmapkeys** program will find out whether the terminal is a scancode device or not. The lines in this section can have two different formats. One format will be used to describe what the values of the function keys must be. The other format describes the translation of scancodes into a byte or an escape sequence. No specific order is required.

Function keys
Here is an example of a line defining a string for a function key:

```
F13 'd''a''t''e''0      # Shift F1 is the date command
```

The numbering convention of the function keys is described in a previous section. Currently, the use of quoted strings such as *"date\n"* is not supported.

Scancodes
Specifying how to translate a scancode is a more complex task. The general format of such a line is:

scancode normal shift alt shiftalt flags

scancode should list the hexadecimal representation of a scancode generated by a key (unquoted). How keys correspond with scancodes can be found in **keyboard**(7D).

normal, shift, alt and **shiftalt** are character representations in one of the formats described throughout this document, optionally followed by one of the following special keywords:

|C This indicates that the key is influenced by the Ctrl key.

|N This indicates that Esc N should preceed the specified character.

|O This indicates that Esc O should preceed the specified character.

|[This indicates that Esc [should preceed the specified character.

The **normal** field defines how the scancode is translated when no other key is pressed, the **shift** field defines the translation for when the Shift key is used simultaneously, the **alt** field specifies what to do when the Alt key is pressed together with this and the **shiftalt** field contains the information on what to generate when both the Shift and Alt keys are pressed.

All five fields must be filled in. When no translation is requested (that is, the current active translation does not need to be changed) a dash (–) can be used. The sixth field is optional. This field can contain the special keyword CAPS or NUM or both, to indicate whether or not the Caps Lock key or Num Lock key status have any effect. Here is a sample line that describes the default translation for the **'Q'** key:

```
0x10 'q'|C  'Q'|C  'q'|N  'Q'|N  CAPS
```

If the normal or shift field is filled out for a scancode that represents a function key, a self-explanatory message will be produced and that translation information will be ignored.

A more detailed example of a **scancodes** section is:

```
scancodes:
# the w key
0x11 'w'|C  'W'|C  'w'|N  'W'|N  CAPS
# left square bracket and curly brace key
# control shift [ does not generate anything (no C flag)
0x1a '['|C  '{'   '['|N  '{'|N
# 9 on numeric keypad
0x49 'V'|[  '9'   '9'|N  '9'|N  NUM
F13 'd''a''t''e''0 # SHIFT F1
```

More complete examples of *mapfile*s can be found in the **/usr/share/lib/keyboards** directory.

OPTIONS

–f *mapfile* Installs the contents of the file *mapfile* and sets the corresponding mapping as supported by the console driver. The layout of the *mapfile* and the supported functionality are described below.

–n Disables and dismantles the current keyboard extended mapping. The **–f** option must be used to re-install the keyboard extended mapping.

–g Displays the current mappings and keyboard extended mapping (if one is installed) in hex values (see **/usr/include/sys/emap.h**). This option is mainly used for debugging purposes.

–d and **–e** **–d** temporarily disables the compose key and deadkey sequences if the keyboard extended mapping is installed. The keyboard extended mapping can be enabled again by using the **–e** option (or it can be re-installed by using the **–f** option).

FILES

/usr/share/lib/keyboards/8859/∗
sample mapfiles to be used in conjunction with ISO-8859-1 fonts (see **loadfont**(1))

/usr/share/lib/keyboards/437/∗
sample mapfiles to be used in conjunction with IBM 437 fonts (see **loadfont**(1))

SEE ALSO

loadfont(1)

NOTES

The default keyboard mappings on the system are those of the ISO 8859-1 codeset. The optional IBM DOS 437 codeset is supported *only* at internationalization level 1. That is, if you choose to download keyboard mappings of the optional IBM DOS 437 codeset, there will be no support for non-standard U.S. date, time, currency, numbers, unit, and collation. There will be no support for non-English message and text presentation, and no multi-byte character support. Therefore, non-Windows users should only use IBM DOS 437 codeset in the default C locale.