Convergent Technologies

ENGINEERING UPDATE
for
CTAM Window and Menu Manager

Revised (June 1988) B-09-01409-01-B

Contents

Using Forms and Menus

Introduction

This document is targeted for an end user of the DPL Forms and Menu system, part of the Convergent Terminal Access Method (CTAM) package on an S/MT system. This guide describes the keystroke sequences available to users of applications displaying forms and menus. After reading this document, you should be able to perform tasks such as selecting items from menus, filling in edit fields, and handling minor error situations.

Since DPL applications displaying forms and menus run on many terminal types (and therefore, many different keyboards), this guide may not describe the keys that exactly match your terminal. Wherever possible, a set of key names from various terminals are described. The primary keynames given are those on the PT/GT terminal.

Some keystroke sequences make use of *control codes*. These are normal keys on your keyboard that are pressed while you hold down the **Control** key (or **Code** key on a PT/GT terminal). For example, on many terminals, the keystroke sequence used to ask the system for help is **Control–h**, that is, the letter 'h' is struck while the control key is held down; this is documented in this manual as **^h**. Whenever you see a preceding caret (^) in a keystroke sequence, interpret the sequence as a control code.

Moving Within Forms and Menus

A *form* is a display containing *fields* of various types. Fields vary in shape and size, depending on the number and format of the items inside. Some fields take up the whole screen, others take up enough space for only a few characters. Depending on the type of field, items in the field can be selected and edited, or selected but not edited, or if the field is for viewing only, neither selected nor edited.

A *menu* differs from a form in that it consists of only one field, and it may or may not take up the entire screen; a menu is actually a field of selectable, non-editable items. In other words, you can choose an item on a menu list, but you cannot change any of the items on the list. Some menus allow you to make more than one selection; when this is true, the screen provides instructions on how to make your choices.

In the example form below, the user is asked to make selections in three fields: to select an entree, to fill in a wine choice, and to select a dessert.

```
Select an entree:  * Beef
                     Chicken
                     Fish

Enter your wine selection: _____________________

Select a dessert:  * Pie
                     Cake
                     Fruit


Make your selections and press 'Go' to execute,
or press 'Cancel' to exit the form without executing.
```

The *arrow keys* (**Up**, **Down**, **Forward**, and **Back**) and the **Tab** and **Return** keys are used to move the *cursor* and make selections on the screen. Fields with more than one item often contain preselected or *default* values that can be changed by moving the cursor to another item in the field. Currently selected items are represented either by a highlighted bar or by an asterisk (*) that the system places to the left of selected items.

In the example form, you would move the cursor within the entree field (using **Up** and **Down**) to make an entree selection. To move the cursor to the wine field, you would use the **Tab** key. You would then type a wine name and press **Return**. (As with the **Tab** key, **Return** moves the cursor to the next field.) You would then make your dessert selection (using **Up** and **Down**).

To indicate that you are satisfied with your selection and that you wish it to be processed by the system, you would press the **Go** key, also called **Do**, or **Linefeed** on other terminals. Some applications ask you to press the **Finish** key (also entered as **^d** on some terminals) when you have completed filling out the form. (Recall that **^d** means that you hold the **Control** key down while you strike the letter 'd'.)

The **Cancel** key (also entered as **^x** on some terminals) allows you to exit a form or menu without executing the currently selected items.

The entree and dessert fields are examples of *menu fields*, which contain selectable but noneditable items. The wine field is an example of an *edit field*, which can be both selected and edited. The prompt line at the bottom is an example of a *text field*, which cannot be selected or edited.

In edit fields, the **Backspace** key deletes the character to the left of the cursor. The **Forward** and the **Back** keys move the cursor without deleting existing text. The **Delete** key deletes the character at the current cursor position. To insert text in the middle of what you have already entered, move the cursor to where you wish to insert and enter the text.

Note that the *arrow keys* allow you to move the cursor both between and within fields. For example, the **Down** key will move the cursor from "Beef" to "Chicken" to "Fish", then down to the wine field, then down to "Pie" to "Cake" to "Fruit", and then it will cycle through the bottom most field. Similarly, the **Up** key moves the cursor up through the fields on a form and then cycles through the upper most field.

To move to another field without changing the selection in a menu field, move the cursor using **Tab** or **Return**.

For more advanced users, a shortcut is available for selecting items in a menu field. Instead of striking the arrow keys a number of times, you may enter the first few letters of the choice that you desire, and the cursor will move to that choice. For the above example, if the cursor is located in the entree field, entering the letter 'F' will move the cursor to 'Fish'.

Some fields contain too many items to fit on the screen. In this case, the system displays a *scroll bar* to the right of the menu or text field to indicate that there are more items than those shown. The scroll bar is a vertical highlighted bar with arrows indicating your relative position on the list of items. To see items that are off the screen, use the down arrow key to force the menu to scroll.

Some menu fields allow you to make more than one selection. If you want to select more than one item, you must use the **Mark** key (labelled **Select** on many other keyboards) to indicate your choices before pressing **Go**. Usually when you press **Mark**, an asterisk appears next to your choice. To 'unmark' something if you've changed your mind on a selection, press **Mark** on that item a second time.

Due to occasional hardware or software problems, the characters on the screen may be displayed improperly, such as broken lines around a menu, or unreadable characters on the screen. The ^l control sequence instructs the system to repaint the screen. If the screen is still unreadable, you should report the problem to your system administrator.

On very rare occasions, the entire screen may freeze up, or the system may appear to ignore your input. For advanced users, the ^\ sequence instructs the system that you would like to exit the application that you are running. The system displays a confirmation form before exiting. Note that this is a potentially dangerous operation and should be used only as a last resort.

CTAM Installation Guide

Introduction

This document describes the various components of the CTAM window system and their places in the CTIX file system. It is intended to be used by someone adding support for a new terminal or anyone interested in customizing the operation of CTAM based products.

CTAM Window Manager Files

The CTAM window manager is a CTAM based application program that works in conjunction with a loadable software device driver called the *wxt* driver. In order for the window manager to function the driver must have been loaded and there must be special files in **/dev** that correspond to it. Most systems will want to have the driver load automatically during boot time and the installation process causes this to happen by modifying files in **/etc**.

/etc/lddrv/wxt.o

This is the driver object code. This driver may be loaded and unloaded manually with the **lddrv(1)** command.

/etc/master

The loadable driver must have an entry in **/etc/master** before it can be loaded. The installation process will create an entry in this file if none already exists.

/etc/drvload

During system startup this shell script is executed to load and start any loadable device drivers used by the system. The installation process will append a line to this file to cause the wxt device driver to be automatically loaded each time the system is re-booted.

/dev/window

/dev/window is a special file that is used to create new windows by the CTAM library when running under the CTAM window manager. When a process opens **/dev/window** the open is re-routed to an unused window sub-device described below. This special file has the major number of the wxt driver and minor number zero. Its permissions should allow reading and writing by all users.

/dev/wxt/w###

The window special files are kept in the **wxt** sub-directory of **/dev**. Each file is named by a "w" followed by three decimal digits representing the device's minor number. Only files named in this way will work. The installation process automatically creates 255 of these files, the maximum number.

Language Dependant Files

There are two files that need to be changed when nationalizing CTAM for use outside the U.S. These files are stored in a sub-directory of **/usr/lib/ctam** whose name is controlled by the LANG environment variable. If the LANG environment variable is not

defined, CTAM applications will default to **english_usa**.

/usr/lib/ctam/english_usa/ctwm.rf

All of the prompts used by the CTAM window manager are stored in this file.

/usr/lib/ctam/english_usa/dpl.rf

This file contains messages and labels common to all DPL based applications.

Terminal Description Files

The CTAM windowing system has been designed to be extensible to work with terminals not directly supported in the released product. Providing support for a new terminal involves adding a number of files to the system that describe various aspects of the terminal for various applications. There are two basic ways in which terminal descriptions are used, the first is by CTAM itself, and the second is by applications running within CTAM windows.

CTAM applications are written to operate a virtual ANSI X3.64 style terminal. All output from an application running within a CTAM window is scanned for escape sequences and other controls that are converted to whatever is needed to get the actual terminal's screen updated. However, input from the actual terminal's keyboard is passed to the application un-touched.

/usr/lib/terminfo/?/*

Two different terminfo files need to be added for every terminal used with CTAM. First, the terminal must have a standard terminfo description (see **terminfo(4)** for a description of terminfo files). Second, a description of the terminal with the name "ctam" appended to it must be added. The purpose of this second file is to enable non-CTAM applications to work within windows. The second

description consists of the keyboard definitions of the standard terminfo description and the output sequences of the ctam description.

/usr/lib/ctam/kbmaps/*.kb

Each terminal used with CTAM should have a keyboard description file. The keyboard description file is an ASCII file consisting of three columns. The first column contains the name of a CTAM "meta-key". A complete list of meta-key names is contained in the file **/usr/include/kcodes.h**. The second column specifies what the terminal is going to send when the key is pressed. Control characters can be specified using the same kinds of textual equivalents as in terminfo *.ti* files. The third column is optional and contains the name the key will be referred to in prompts to the user by some applications. Not all keys need be defined, only those which have an obvious counterpart on the new terminal's keyboard. Meta-keys that are not defined will have a default multi-key

/usr/lib/ctam/fonts/*.ft

Terminals with additional fonts beyond standard ASCII should have a font description file. The font description file is similar in format to a **terminfo** *.ti* file. A keyword is followed by an equals sign "=" which is in turn followed by a definition string and finally terminated by a comma or end-of-file.

/etc/CTWMtermcap

This file is used by non-CTAM, termcap-based applications running inside a CTAM window. The file contains a description for the generic CTAM virtual terminal and keyboard descriptions for several popular terminals.

/etc/termcap

All terminals used on a system should have an entry in **/etc/termcap** although CTAM applications will not actually use **/etc/termcap**. The reason for this is so that **tset(1)** will correctly set the **TERM** environment variable when the user logs in.

Example

In this example, a new terminal will be added to the system. The terminal will be a MicroTerm Act-IVA.

1. Create a *terminfo* entry for the terminal. In this example, there happens to already be a description for the terminal called "act4". The compiled entry for this terminal resides in */usr/lib/terminfo/a/act4*.

2. Check operation of the window manager. Once the *terminfo* entry is in place, the CTAM Window Manager should work. A simple test is to type:

   TERM=act4 export TERM
   ctwm

   The window manager should recognize the terminal and bring up a default window running a shell. Check to see just how well everything works. Are the window borders solid or broken lines? Do any of the arrow keys or function keys work? If the screen gets wrecked then the *terminfo* description needs work, if the borders or function keys don't work but the screen is otherwise ok, then further steps need to be taken.

3. Create a keyboard description file for the terminal. Make a new file in */usr/lib/ctam/kbmaps* for the terminal and name it *act4.kb* (i.e. name of terminal followed by ".kb"). At a minimum, put entries for the terminal's arrow keys and function keys in the file. Check the success of the new keyboard description file by re-running the window manager.

4. Create a font description file for the terminal. If the terminal has a graphics font or is somehow capable of drawing solid lines, then a font description file should be added. In this example, it would be called */usr/lib/ctam/fonts/act4.ft*. The most useful font to add is the **decgraph** entry. Again, check the

success of the new font description file by re-running the window manager.

5. At this point, all CTAM applications (this includes WGS) should be fully functional. If non-CTAM applications are to be used under the window manager, then continue.

6. For *terminfo* based applications to properly recognize a terminal's function and arrow keys under the window manager, a second terminfo description must be added. The second description contains the keyboard entries of the primary terminfo description and the output entries of the "ctam" terminfo description. The ctam description is in the file */usr/lib/terminfo/c/ctam*. The new file should be called */usr/lib/terminfo/a/act4ctam*.

7. For *termcap* based applications to properly recognize a terminal's function and arrow keys under the window manager, an entry must be made in */etc/CTWMtermcap*. This is considerably easier than adding the second terminfo description since the termcap file is an editable ASCII file. Append to the file an entry named *act4ctam* that describes the terminal's function and arrow keys followed with a "tc=ctam" entry that will cause the rest of the entry to be the same as the "ctam" termcap entry.

NAME

ctwm – CTAM Window Manager

SYNOPSIS

exec ctwm [-r visible rows] [-c visible columns] [-x start column] [-y start row] [-h height] [-w width] [-e switch key] [-I command file] [-b] [-f] [-g] [-o] [-p] [-s] [-t] [initial shell]

wexec [-r visible rows] [-c visible columns] [-x start column] [-y start row] [-h height] [-w width] [-b] [-f] [-g] [-p] [command]

wconfig [-u max. user windows] [-s max. super user windows]

DESCRIPTION

Ctwm is the CTAM window manager which enables multiple applications to run simultaneously on a terminal in multiple windows transparent to the application. With *ctwm* the output of several programs is coordinated for display on the user's terminal such that each application is confined to a particular rectangular region or "window" on the screen. Each window functions as an entire virtual display screen distinct from the other windows. Output sent to the screen by an application program is clipped by the windowing manager to fit in its window's *viewport*. The viewport size is defined by the number of rows and columns visible to the user between the window borders when the window is un-obscured by other windows. The size and placement of windows on the screen is arbitrary and completely under user control.

Application programs are often written to take advantage of an entire screen. CTAM supports full screen *pads*, where a pad is the screen area into which the viewport allows the user to see. Commands are available to scroll the pad, or change the viewport size to afford a full view of the contents of the pad. Full screen pads are stored by CTAM for every screen. As a result, programs that are written to use a whole screen work correctly unchanged under the windowing system. The CTAM windowing system uses the terminfo(4) database to determine individual terminal characteristics, as well as the appropriate file under the directories /usr/lib/ctam/fonts and /usr/lib/ctam/kbmaps. (See FONTS (4w) and KBMAPS (4w).)

The window manager supports features to manipulate windows. By pressing Control-Z (Code-Z on a Convergent PT or GT terminal), the user enters a mode in which all of his or her commands are directed to the window manager. The user can change Control-Z to any other character by setting the *switch key* option. In most cases, it is necessary to enter Control-D (Finish on a Convergent PT or GT) to exit the window manager and return control to the application running in the topmost window. When in window manager mode, the user's function keys are labeled to support the following features:

CREATE Create a window. By selecting this function key, the windowing system will create a new window with the user defined window size. The shell defined by *initial shell* is spawned into that window.

SWITCH Switch topmost window. By selecting this function key, the user enters a mode where the arrow keys on the terminal are used to select the window to become the new topmost window. It is this window that will receive input from the keyboard. **Down arrow** goes forward

through the existing windows and **Up arrow** goes backward the existing windows in order, selecting in turn, each window as the new topmost window. When the user has selected the window that he or she requires for the new topmost window, he enters Control-D, or return, to return control to the application program running in that window. The user can also specify a particular window by either entering the window number (i.e. 0-9 where 0 is window #10) or the first letter of the window label. Control is automatically returned to the application program running in that window. If two or more windows have the same first letter, the window with the lowest window number will be activated.

MOVE Moves a window. By selecting this function key, the user enters a mode where the arrow keys on the terminal are used to move the current topmost window. The arrow keys can be preceded by a number to move the window more than one slot at a time. When the user has placed the window in the desired location, he or she enters Control-D, or Return, to return control to the application program running in that window. To move the window more than one space at a time, a number followed by an arrow key may be used.

GROW Grows a window. By selecting this function key, the user enters a mode where the arrow keys on the terminal are used to make the topmost window grow. Each arrow key grows the corresponding window border in that direction. Arrow keys may be preceded by a number to grow the window more than one slot at a time. When the user has the window the size he or she wants it, he enters Control-D, or return, to return control to the application program running in that window.

SHRINK Shrinks a window. By selecting this function key, the user enters a mode where the arrow keys on the terminal are used to make the topmost window shrink in size. Each arrow key shrinks the corresponding window border in that direction. The arrow key may be preceded with a number to shrink the window more than one slot at a time. When the user has the window the size he or she wants it, he enters Control-D, or return, to return control to the application program running in that window.

SCROLL Scrolls the pad. By selecting this function key, the user enters a mode where the arrow keys on the terminal are used to scroll the display in the viewport. The user enters Control-D, or return, to return control to the application program running in that window. To scroll the pad more than one space at a time, a number followed by an arrow key may be used.

MAX/PRE Size the window to the maximum or previous size. By selecting this function key, the user enters a mode where the window size is changed to the maximum or the previous size. The user enters Control-D, or return, to return control to the application program running in that window.

MENU Displays a menu of all the existing window labels. The user selects the desired window and press return to return control to the application

program running in that window.

TOPWIN Switch to top controlling window. This function replaces the CREATE *function if the -t option is present.* This is useful in the case where the user is only allowed to start new activities through a particular controlling window. This feature works well with the *-s* option (see below) in hiding the operating system from the naive user.

To refresh the screen's current contents, press Control-L (or Code-L). Control-C (or Code-C) pops up a menu of things to do on the screen.

To enter a shell command, press ! followed by the command and press return. The command will be echoed on the command line and the command will be executed in a new window. After the command is finished, the user is prompt to acknowledge this. Control is returned to the next active window. This feature is disabled if the *-s* option is present.

Startup Options

The *initial shell* is the name of the shell that the user would like started up in all new windows created with the "CREATE" key. *Vrows* and *vcols* tell the windowing system how large to make new windows. *Start column* and *start row* describe the initial column and row position of the upper left corner of the first window. The upper left corner of the screen is at position (0,0).

If no sizing information is provided, *ctwm* defaults the window size to 22 rows by 78 columns, not counting the border characters. If *vrows* and/or *vcols* are set, but not the initial positions, when the windowing system creates a new window, it looks for a free area on the screen.

The *-b* option, if present, makes all windows borderless windows.

The *-f* option, if present, makes all windows fixed-size windows.

The *-o* option, if present, disallows the user from creating windows.

The *-s* option, if present, disallows the user from entering a shell command on the command line.

The *-t* option, if present, makes the resulting window from the corresponding shell command the top controlling window.

The *-p* option makes the pad and the viewport the same size.

The *-g* option prevents the window from scrolling to track the cursor.

If an application requires that the *pad* be a size other than that of the physical screen, the *height* and *width* options can be used to set the default. The *wtty* command can be used to change this from the shell. Programmatically, this can be changed by means of the *WSetArgs* CTAM call.

The user can specify a list of commands to be executed in different windows when the windowing system first starts up by specifying the command list file name with the *command file* option. The format of the command list is:

[-r #] [-c #] [-h #] [-w #] [-x #] [-y #]

```
[-p] [-b] [-f] [-l] shell command [options]
```

All the options correspond to the *ctwm* options. For example,

```
-r 4 -c 10 date; pwd; exec $SHELL
$SHELL
```

will create a 4 by 10 window executing a shell and a full size window executing another shell.

The windowing system supports a user-configurable background character to occupy the parts of the screen that do not contain a window. That character is defined by the optional environment variables **CTWM_BG** and **CTWM_ATTR**. For example, the following shell commands set the character "." as the background character, and run the window manager, starting the C-shell in a 10 by 20 window:

```
$ CTWM_BG=. ; export CTWM_BG
$ exec ctwm -h 10 -w 20 csh
```

If the **CTWM_BG** environment variable is not set, the background character defaults to blank. The environment variable **CTWM_ATTR** controls the attribute with which the background is displayed. The attribute is defined by giving the parameters to the **SGR** escape sequence (see *escape*(7W)) to be used with **CTAM_BG**. For instance, setting **CTWM_ATTR** equal to "7" would cause the background to be displayed in reverse video.

The windowing system also supports a user-configurable forms and menu file. The default file is */usr/lib/ctam/english_usa/ctwm.rf*. The user can specify a customized file by setting the environment variable **CTWM_FORM**.

It is required that you start *ctwm* by using the shell's *exec* function (see example). When the window manager is run, the user's **TERM** variable must be correctly set to the terminal on which the windowing system will run. Note that when the *initial shell* is spawned by the windowing system, the **TERM** variable is changed to reflect the requirements of *ctwm*. For example, the **TERM** variable *pt* is changed to *ptctam*. This must remain set in this way for correct operation of the windowing system. In addition, *ctwm* sets the **TERMCAP** variable to **CTWMtermcap**.

Window Signal

If an application running under the window manager wishes to know when any of its windows is selected or re-sized by the user via the window manager, it should include a signal catching routine for **SIGWIND** (signal number 20). This signal will be sent whenever a window becomes active (selected) or is re-sized by the window manager. If a program has multiple windows open it should call *WGetSelect* to see which, if any, of its windows is the active window. If the program has windows that can be re-sized by the user (windows without the **FIXEDSIZE** flag set in their window status structure), then the program should call *WGetArgs* on each window to see what actually happened.

SEE ALSO

wtty(1W), terminfo(4), fonts(4W), kbmaps(4W).

WARNINGS

Ctwm is designed to be run from the host machine. Applications to be run over a network are supported in windows.

NAME

dplrun – Interpret a resource file

SYNOPSIS

dplrun [-s StartingForm] file.rf [...file2.rf...] [-c opts...]

DESCRIPTION

dplrun interprets a **resource file**, executing the menu and form information in the file. *file.rf* is the name of the resource file. The **-s** option provides the interpreter with the name of the starting form to display. If not used, the interpreter looks for the form named **mainform**.

The **-c** option allows the user to set the values of $1, $2, $3, etc., to be used by the starting form.

EXAMPLES

```
dplrun -s StartingForm MyFile.rf

dplrun MyFile.rf -c Hello World
```

NAME

wtty – set window configuration for ctwm

SYNOPSIS

wtty [-height height] [-width width] [-vrows vrows] [-vcols vcols] [-begx begx] [-begy begy] [-border] [-fixedsize] [-padwin] [-track]

DESCRIPTION

This command should be run from the shell inside a CTAM window. It is used to provide information about the window in which it is run or to alter the parameters for the window.

If given no parameters, *wtty* reports the beginning X and Y coordinates of the left hand corner of the window (*begx*, *begy*), the number of rows and columns in the viewport (*vrows*, *vcols*), the height and width of the pad (*height*, *width*), whether the window has borders, and whether the pad and the viewport are the same size. See the description of *ctwm* for a discussion on pads and viewports.

Height and *width* affect the size of the pad. *Vrows* and *vcols* affect the size of the viewport. *Begx* and *begy* affect the beginning coordinates: the upper left corner of the window. The *-border* flag causes the window to be displayed without borders surrounding it. The *-padwin* flag causes the pad and viewport to be "locked" together and always be equal. The *-fixedsize* flag prohibits the user from changing the window size with the window manager. The *-track* flag enables scrolling of the window to track the cursor.

SEE ALSO

ctwm(1W), WGetArgs(3W).

NAME

fonts – CTAM Font mapping files

SYNOPSIS

/usr/lib/ctam/fonts/*.ft

DESCRIPTION

CTAM employs a font description database in order to map the virtual font set available to an application to the actual fonts available on the terminal. If no fonts beyond ASCII are available in the terminal then no font description file is needed. However, if the terminal is capable of displaying different fonts then these fonts need to be described with a font description file.

It is assumed that the terminal has one or more alternate fonts that may be selected and de-selected by use of multi-character sequences. These character sets are referred to as "alternate character sets". CTAM allows up to three different alternate character sets to be used to describe fonts. In order to correctly map CTAM's idea of what characters are going to appear on the screen it is necessary to establish a mapping between CTAM's virtual fonts and the terminal's real fonts. First, the sequences to switch between the terminal's alternate character sets must be specified. For example, if the terminal has a special graphics font that is selected by the sequence (1Bh, 65h) and de-selected by the sequence (1Bh, 66h) then the font file would contain the following:

```
smacs2=Ef, rmacs2=Eg,
```

The font mapping for each CTAM virtual character set is specified by a string containing sequences of three tuples. The first character in each tuple gives the position in the CTAM virtual character set by the equivalent ASCII character. The second character gives the position in the terminal's font of the desired physical character also by the equivalent ASCII character. The third character specifies the alternate character set number to be used when display the character or a '~' to indicate just that the high order bit should be set when displaying the character. Alternatively, if the terminal has a font that exactly matches a particular CTAM virtual font then that may be specified by the name of the virtual font, followed by an equals, followed by a single character representing the alternate character set that must be used to display that font.

Names of virtual fonts include: usascii, ukascii, decmulti, decgraph, ctgraph, ctline, user1, user2, and user3. Alternate character set 1 should be specified in the terminal's terminfo description file using the smacs and rmacs capabilities. Two additional alternate character sequences may be defined in the font map file using smacs2, rmacs2, smacs3, and rmacs3.

EXAMPLE

The Fortune terminal has an alternate character set containing many of the same symbols as the CTAM CT Graphics character set. In the following example, nine of these special characters are mapped from the Fortune Systems Graphics Character Set onto the CT Graphics virtual font:

```
smacs=^N, rmacs=^O,
ctgraph=X81 >>1 <01 $41 &51 =11 |<1 ??1 ~v1,
```

The mapping string is made up of nine three-tuples each specifying a single character. White space in the mapping string is ignored. The first three-tuple states

that the 'X' position of the CT Graphics Character Set (58 hex) is displayed by outputting an '8' (38 hex) when the terminal is in alternate character set 1.

FILES

/usr/lib/ctam/fonts/*.ft - Terminal font description database

SEE ALSO

terminfo(4).

For descriptions of the **decgraph**, and **decmulti** character sets refer to the *VT-220 Programmer's Reference Manual EK-VT220-RM* or equivalent. For descriptions of **ctgraph**, and **ctline** character sets refer to the *Convergent Programmable Terminal Programmer's Guide*, figures A-3, and A-2.

NAME

kbmaps – CTAM keyboard mapping files

SYNOPSIS

/usr/lib/ctam/kbmaps/*.kb

DESCRIPTION

CTAM programs access files in the directory */usr/lib/ctam/kbmaps* (or a directory named by the **KBMAP** environment variable) to determine information about a terminal's keyboard beyond what is described by *terminfo*(4). The information in the terminal's keyboard description file supersedes whatever information is specified in *terminfo*. The files in */usr/lib/ctam/kbmaps* consist of lines of three fields each. The first field specifies the internal name of a key. A complete list of valid internal names is contained in */usr/include/kcodes.h*. The second field specifies what the terminal sends when that key is pressed. The third field is optional and if present gives the keycap label for the key.

Key Semantics

The semantics of CTAM metakeys varies from one application to another. However, since the internal names of some metakeys do not accurately reflect their common usages, a list of basic keys and their meaning is presented here:

RollUp	Scroll down
RollDn	Scroll up
Page	Next page
s_Page	Previous page
Forward	Right arrow or character right
Back	Left arrow or character left
Up	Up arrow or line up
Down	Down arrow or line down
Home	Beginning of page
s_Home	End of page
Beg	Beginning of document
End	End of document
Next	Shift right arrow, next word
Prev	Shift left arrow, previous word
s_Forward	Control right arrow, full scroll right
s_Back	Control left arrow, full scroll left
ClearLine	Erase field
DleteChar	Character delete
InputMode	Toggle insert/replace mode

EXAMPLE

The following example is from a keyboard mapping file for a Fortune terminal.

```
#
# Fortune keyboard description file
#
F1                      ^Aa^MF1
F2                      ^Ab^MF2
F3                      ^Ac^MF3
F4                      ^Ad^MF4
F5                      ^Ae^MF5
F6                      ^Af^MF6
```

F7	^Ag^MF7
F8	^Ah^MF8
F9	^Al^MF9
F10	^Ak^MF10
Help	^A@^MHelp
s_Page	^As^MPrevScrn
Beg	^AS^Ms-PrevScrn
Page	^Au^MNextScrn
End	^AU^Ms-NextScrn
Home	^AX^Ms-Up
s_Home	^AY^Ms-Down
Next	^AZ^Ms-Right
Prev	^AW^Ms-Left
Enter	^Aq^MExecute
InputMode	^Ar^MInsert
DleteChar	^At^MDelete

WARNINGS

It is important to avoid ambiguities in keyboard definitions. If one key sequence is a sub-set of another key sequence, the shorter of the two will always prevail. A system integrator adding support for a new terminal should watch out for this potential problem as CTAM does not check.

SEE ALSO

fonts(4W), terminfo(4).

NAME

escape – window escape codes

DESCRIPTION

CTAM windows emulate an extended ANSI X3.64 style terminal where special sequences of characters embedded in the output stream control certain aspects of the window. These aspects include character display attributes like reverse video and underlining as well as scrolling and erasing. Sequences of special characters written to the window via *wprintf* (3W), *wputc* (3W), and *wputs* (3W) are interpreted by CTAM along with normal text.

There are three broad categories of control sequences: C0 controls, C1 controls, and multiple character. C0 control sequences are the familiar ASCII controls such as 0Dh (carriage return) and 0Ah (linefeed). C1 control sequences may be sent in two ways, as a single eight-bit value or as the ASCII escape code 1Bh followed by a second character. Multiple character sequences all begin with the C1 control called the Control Sequence Introducer. The CSI control code may expressed as the single eight bit value 9Bh, or as the two character sequence 1Bh 5Bh (Escape [). This type of control sequence is used for more complex operations.

C0 Controls

Name	Sequence	Description
NUL	00h	Null (ignored)
BEL	07h	Sound Bell
BS	08h	Backspace if col > 1
HT	09h	Horizontal tab
LF	0Ah	Linefeed; scroll up at bottom of scroll region.
VT	0Bh	Vertical tab; down one or scroll up at bottom of scroll region.
FF	0Ch	Form Feed; same as VT
CR	0Dh	Carriage Return; cursor moves to column 1
SO	0Eh	Shift out; selects G1 character set for GL
SI	0Fh	Shift In; selects G0 character set for GL

C1 Controls

Name	Sequence	Description
IND	84h or Esc D	Index (same as linefeed)
HTS	88h or Esc H	Horizontal tab set
RI	8Dh or Esc M	Reverse Index; scroll down in row 1
SS2	8Eh or Esc N	Single shift G2 into GL for the next character
SS3	8Fh or Esc O	Single shift G3 into GL for the next character
NEL	85h or Esc E	New Line; move to column 1 of next line
CSI	9Bh or Esc [	Control Sequence Introducer; see below
SC	Esc 7	Save cursor position and cursor attributes
RC	Esc 8	Restore cursor position and attributes.
LS1R	Esc ~	Lock shift G1 into GR
LS2	Esc n	Lock shift G2 into GL
LS2R	Esc }	Lock shift G2 into GR
LS3	Esc o	Lock shift G3 into GL
LS3R	Esc |	Lock shift G3 into GR

Multiple Character Sequences

Name	Sequence	Description
CUP	CSI Ps1 ; Ps2 H	Move cursor to column Ps1, row Ps2
CUU	CSI Pn A	Move cursor up Pn lines
CUD	CSI Pn B	Move cursor down Pn lines
CUF	CSI Pn C	Move cursor forward Pn columns
CUB	CSI Pn D	Move cursor back Pn columns
CNL	CSI E	Move cursor to column 1 of next line
CPL	CSI F	Move cursor to column 1 of previous line
SU	CSI Pn S	Scroll up Pn lines
SD	CSI Pn T	Scroll down Pn lines
DCH	CSI Pn P	Delete Pn positions
ICH	CSI Pn @	Insert Pn positions
ECH	CSI Pn X	Erase (change to space) next Pn positions
DL	CSI Pn M	Delete Pn lines
IL	CSI Pn L	Insert Pn lines
EL0	CSI 0 K	Erase cursor to end of line
EL1	CSI 1 K	Erase beginning of line to cursor
EL2	CSI 2 K	Erase entire line
ED0	CSI 0 J	Erase cursor to end of display
ED1	CSI 1 J	Erase beginning of display to cursor
ED2	CSI 2 J	Erase entire display
SGR0	CSI 0 m	Set all attributes to normal
SGR1	CSI 1 m	Select bold
SGR2	CSI 2 m	Select dim
SGR4	CSI 4 m	Select underline
SGR7	CSI 7 m	Select reverse
SGR9	CSI 9 m	Select struck out
SGR21	CSI 21 m	Turn off bold
SGR22	CSI 22 m	Turn off dim
SGR24	CSI 24 m	Turn off underlining
SGR27	CSI 27 m	Turn off reverse
SGR29	CSI 29 m	Turn off struck out
TBC0	CSI 0 g	Remove horizontal tab stop at current position
TBC3	CSI 3 g	Remove all horizontal tab stops
CSR	CSI Ps1;Ps2 r	Set scroll region

DSR	CSI n	Device status report
CTSLP0	CSI = 0;Ps2 @	Move to prompt line, column Ps2 (see CTSLN)
CTSLP1	CSI = 1;Ps2 @	Move to tag line, column Ps2 (see CTSLN)
CTSLP2	CSI = 2;Ps2 @	Move to SLK line, column Ps2 (see CTSLN)
CTSLP3	CSI = 3;Ps2 @	Move to command line, column Ps2 (see CTSLN)
CTSLN	CSI = Ps1 q	Set the number of active noise lines (Before any special line positions (CTSLP0-3) can be used, CTSLN must be used.)
CTVIS0	CSI = 0 C	Make cursor visible
CTVIS1	CSI = 1 C	Make cursor invisible
CTMF	CSI = Ps1;.. R	Map fonts Ps1... to G0...
CTSU	CSI = Ps1;Ps2;Pn S	Scroll lines Ps1 through Ps2 up Pn lines
CTSD	CSI = Ps1;Ps2;Pn T	Scroll lines Ps1 through Ps2 down Pn lines
CTWN	CSI = W	Write window number
CTDSR	CSI = b	Device status report
CTSGR	CSI = Ps1;Ps2m	Select Ps1 = on mask; Ps2 = off mask
CTSM2	CSI = 2 h	Clear and enable window label
CTSM7	CSI = 7 h	Save cursor (same as SC)
CTTRON	CSI = 3 h	Enable cursor tracking
CTTROFF	CSI = 3 l	Disable cursor tracking
CTRM2	CSI = 2 l	Disable window label
CTRM7	CSI = 7 l	Restore cursor (same as RC)
CTRESET	CSI = p	Reset window to initial modes
CTSD	CSI = 0 w	Disable scrolling
CTSE	CSI = 1 w	Enable scrolling
DECCOLM	CSI ? 3 h	Set window width to 132 columns
DECOM	CSI ? 6 h	Set origin (1,1) to be top of scroll region
DECAWM	CSI ? 7 h	Enable autowrap at column 80
DECTCEM	CSI ? 25 h	Same as CTVIS0
DECRM3	CSI ? 3 l	Set window width to 80 columns
DECRM6	CSI ? 6 l	Set origin to be top of screen
DECRM7	CSI ? 7 l	Disable autowrap
DECRM25	CSI ? 25 l	Same as CTVIS1
DECDSR	CSI ? n	Device status report
DSG0	Esc (*F*	
DSG1	Esc) *F*	
DSG2	Esc * *F*	
DSG3	Esc + *F*	

Designate character set G0, G1, G2, or G3 as font *F* where *F* is:
'A' for UK ASCII, 'B' for US ASCII, '0' for DEC special graphics, '<' for DEC multinational, '=' for PT special graphics, '2' for user font 1, '3' for user font 2, or '4' for user font 3.

FILES

/usr/include/ctam.h