Laser-Scan Ltd.

LITES2

VAXstation (UIS) Workstation Guide

Issue 2.5 - 6-April-1992

Science Park, Milton Road, Cambridge, England CB4 4FY tel: (0223) 420414

Document "LITES2 - VAXstation (UIS) Workstation Guide"		Category "USER"
Document Issue 1.0	Clarke Brunt	2-Oct-1986
Document Issue 1.1	Ron Russell	13-Feb-1987
.		
.		
.		
Document Issue 2.4	Clarke Brunt	24-Mar-1991
Document Issue 2.5	Clarke Brunt	6-Apr-1992

1 *Introduction*

This document describes the workstation dependent facilities available in the version of LITES2 for DEC VAXstation (UIS) displays (image LITES2UIS.EXE). It is to be read as a supplement to the LITES2 Reference Manual and the LITES2 User's Guide.

2 *Display*

LITES2 may be run from a terminal emulating window on the VAXstation screen, or from a separate terminal. The graphic displays appear in windows on the VAXstation screen, which is accessed on logical name SYS$WORKSTATION (which is normally set up system wide). Up to 4 display windows are supported. GRAPHICS must be enabled. Primary and secondary displays are enabled using ENABLE PRIMARY and ENABLE SECONDARY. Both primary and secondary windows support cursor movement, highlighting, and interaction. If both primary and secondary displays are enabled, then it is likely that the SUPPRESS command will be used in order for instance to keep a full map view in one window while zooming in the other. The third and fourth display windows are used by the VIEW command, or the various ANNOTATION and DRAW commands. The attributes (size, shape, number of colours etc.) of all four windows may be set using DISPLAY commands. It may be necessary to restrict the number of colours in each display (using DISPLAY COLOURS, and the mechanism described below) so as not to exceed the maximum number of colours which the VAXstation can display independently. The colours in the primary and secondary windows are taken initially from the colour lookup table on logical name LSL$VAX_COLOUR (see below), and the WORKSTATION COLOUR command affects both windows. Overlays may be created (by WORKSTATION OVERLAY) in the primary and secondary displays - the colours and overlay structure are shared between the two, so for instance creating an overlay in one also creates the same overlay in the other.

ENABLE SEGMENTS is not supported, so all redrawing is performed from the source IFF file.

On an 8 plane system, the LITES2 cursor, and all highlighting of found features etc., is drawn using exclusive or mode, in a primary colour complementary to the underlying colour. Colours 0-63 may be used for the map data (0 is the background).

On a 4 plane system, colours 0-7 may be used for the map. Highlighting also uses these colours in such a way that colour 7 is highlighted in colour 0, 6 in 1, 5 in 2 etc. The colour table should therefore be set up in such a way that the colours are distinct.

On a monochrome system, only colours 0 and 1 are available. In all cases, features specified with colours out of range will be drawn in colour 1.

The number of colours used may be modified to some extent by the use of the WORKSTATION TYPE command.
WORKSTATION TYPE 1 6000 is relevant only to the 8 plane system, and allows the user to define 128 rather than 64 colours. A side effect of this is that highlighting is done in one of the user-defined colours, rather than in a primary colour defined by LITES2. Colour 1 is highlighted as 126, 2 as 125 etc.
WORKSTATION TYPE 1 7000 is relevant to both 4 and 8 plane systems. The number of colours used is then allowed to reach the maximum for the particular number

of planes (16 and 256 respectively), the defaults being 16 and 250. The choice of 250 colours on an 8 plane system allows LITES2 to run concurrently with any terminal emulation windows on the screen without any contention over the colours. If logical name LSL$UIS_MAX_COLOUR is defined to be an integer, then this number of colours will be used instead. This could be used for example to use the full 256 colours, even though this will then result in contention with any terminal windows. If a LITES2 screen menu is in use, then this logical name may be used to restrict LITES2 to 248 colours, in order to avoid any contention over the colours.

The colours used for the picture are defined using a text file on logical name LSL$VAX_COLOUR (if this is set up). See Appendix A for the format of this file. Colours may be changed subsequently using the WORKSTATION COLOUR command.

The cursor may be small or large, and blinking or steady (ENABLE/DISABLE BIG/BLINK).

This version of LITES2 supports display overlays. The number of bit planes available for use as overlays is controlled by the WORKSTATION TYPE command and LSL$UIS_MAX_COLOUR logical name as described above. VAXstations are usually more efficient at displaying images in rows starting at the top of the screen rather than in columns, thus IMAGE CORNER NW and IMAGE DIRECTION CLOCKWISE give the most efficient display. Experimentation with these commands should show the effect, and if necessary, DTI files may be rotated to the best orientation using the MATRIX package module DTIROTATE. The speed of drawing is also affected by the presence of other windows on top of the LITES2 graphics window, and whether the window lies entirely on the screen - again it is worth experimenting by popping the graphics window to the top (DISPLAY POP may be used) while drawing an image to observe whether this gives a significant speed gain.

Note that raster images might not be drawn correctly if both the primary and secondary windows are active simultaneously, especially if the windows are of different sizes. If this is a problem, then use SUPPRESS to cause drawing only into one window at a time.

Two logical names are available to control the way in which drawing on the UIS workstation takes place:

If logical name LSL$UIS_HW_FILL is defined (as anything), then the hardware area filling facilities are used for solid fill areas (by default, close horizontal hatch lines generated by the software are used). Hardware filling may well be faster than the software fill. Hardware fill may be of particular benefit if a UIS metafile is being created (see below), since it will take advantage of the filling facilities of the final output device. By default, filled areas are limited to 8192 points, but this limit, and the number of times a scan line may cut the area, may be altered by defining logical names LSL$FILL_POINTSMAX and LSL$FILL_CUTSMAX respectively.

Logical name LSL$UIS_RESOLUTION may be set to a line width in mm (the default is 0.3, or approximately one pixel). This width will be used for lines with zero width in the FRT, and will be the thickness and separation of hatch lines in solid filled areas if hardware fill is not used. The main purpose of this is that lines with a width less than the screen resolution may be specified if output is to a UIS metafile for final output on a different device.

In order to save a picture in a UIS plot file, define the logical name LSL$UIS_RETAIN_FILENAME - this is the name of the file which will contain the UIS commands (i.e. the metafile). This option is intended more for use with the plotting program FPPUIS than with LITES2. If used with LITES2, then the minimum drawing required to produce the picture should be performed, and the session then terminated, since any cursor movement, editing, redrawing etc. will all be saved in the metafile.

It is important that this logical-name is cleared when UIS output-to-file is complete because display-lists have to be enabled which can have unpleasant side-effects on other programs using UIS.

(NB. A useful VMS utility which operates on UIS files is RENDER - it can translate the UIS commands into several formats eg. SIXEL, PostScript etc - see "MicroVMS WORKSTATION SOFTWARE" manual.)

3 *Interactive devices*

In addition to the keyboard, this version of LITES2 is capable of interpreting commands from a digitising table on a separate serial line, the VAXstation mouse or bitpad, a screen menu, a Laser-Scan button box connected on a separate serial line, and (as a separately licensed option) a KERN DSR photogrammetric plotter with optional KRISS image superimposition system. See the "LITES2 - KERN DSR Workstation Guide" for details of using the DSR.

3.1 *Digitising table*

The digitising table is activated by the commands ENABLE TABLE and ENABLE MONITOR (both are required). The table puck may be programmed using the PUCK command on device 3. The digitising table input is interpreted either using the Table Monitor system, or by reading the table directly. The former allows the table to be set in stream mode, giving smooth cursor tracking.

To use the Table Monitor, a table monitor process must be started, using program STARTMON. If the 'named monitor' option is used, then logical name LSL$MONITOR_TABLE must point to the serial line. In addition, if the table is anything other than a standard ALTEK, then logical table LSL$TABMON_ROUTINE (or LSL$TABMON_ROUTINE_<terminal> for named monitor) must point to a suitable decoding shareable image. This logical name must be available to the table monitor process, and so should be in the group or system logical name table. If stream mode is used, to allow smooth tracking using the lowest numbered button, then the lowest acceptable stream rate above 4 points per second should be used. If set too high, then the table monitor will use large quantities of system resources, if too low, then buttons other than the 'tracking button' will repeat if held down.

If logical name LSL$MONITOR_TABLE is set up, but LITES2 determines that no table monitor process exists, the table will be accessed directly. This does not allow stream mode or smooth tracking.

3.2 *Mouse or bitpad*

The VAXstation is fitted with either a mouse or bitpad to control the windowing system. The commands ENABLE BITPAD or ENABLE BALL activate this device for use by LITES2. LITES2 does not distinguish between a mouse and a bitpad - either command will enable either device. The only difference is that ENABLE BITPAD will allow a menu to be set up on the bitpad.

LITES2 will respond to mouse or bitpad button presses only when the VAXstation cursor is within the graphics window. The LITES2 cursor will follow the VAXstation arrow cursor whenever the lowest numbered button (left mouse button) is pressed or held down within the LITES window. The remaining buttons may be programmed as a puck on device 2 (ENABLE BITPAD) or device 4 (ENABLE BALL). If both BITPAD and BALL are enabled, then the bitpad puck takes precedence.

A menu may be set up on the VAXstation bitpad (if a mouse is fitted, there is little point to setting up a menu unless the menu is attached to the screen). When prompted to set up the menu, any bitpad puck button may be used to digitise the corners. If the tracking button is used, care must be taken not to move while pressing it, or several points will be entered. As usual, the setup may be retained/aborted using the last two buttons on any defined puck (this includes the screen menu). Note that the cursor must be within the graphics window in order to access the bitpad menu. The terminal emulation window and screen menu should be positioned so as not to prevent access to parts of the menu, and care must also be taken if the graphics window is moved.

If a menu is used on the bitpad, then attention should be drawn to the PRIORITY PUCK command. The lowest numbered button is always used for cursor tracking, so for the 4 button puck, 2 other buttons may be given priority so that their puck function is obeyed even if they are pressed over the menu. At least one button must not be given priority, otherwise it will be impossible to access the menu.

Whilst it is possible to set up a tracking area on the bitpad, this is not recommended, since the LITES2 cursor would not move to the position of the VAXstation cursor. In any case, the lowest numbered button tracks anyway.

3.3 *Screen menu*

A facility for a single screen menu is provided within LITES2 using the DEC VWS/UIS windowing facilities. The lowest numbered button (left button on the mouse) must be used on the screen menu.

Before a screen menu can be displayed on the screen, it must be defined as a PUCK on device 1 (the screen). This also defines the total number of boxes in the menu. The boxes of the screen menu can be programmed using MACRO commands as usual.

The size, layout, screen position and title of the screen menu are defined by the DESCRIBE SCREENMENU command. The titling of the boxes is achieved by the DESCRIBE MACRO command.

The menu is displayed on the screen by the ENABLE SCREENMENU command. If this command is given in INITIAL state, then a screen menu window will be created just after the graphics window appears. When a map has been read in, the existence of the screen menu can be controlled by the ENABLE, DISABLE and TOGGLE

SCREENMENU commands.

Like any other Vaxstation window, the menu can be picked up and moved.

3.4 *UISMENUS*

A utility program UISMENUS is supplied which can generate a hierarchical tree of menus for sending commands to LITES2. The utility is described in full in Appendix C. This menu system is most useful when LITES2 is being used as a graphical enquiry tool as part of a captive command procedure, and the menu system is used as the controlling interaction stream.

UISMENUS runs as a separate program from LITES2 but communicates via a mailbox using the auxiliary input to LITES2 referred to by the logical name LSL$LITES2AUX. Before running UISMENUS the initialisation command file must be run:

```
$ @LSL$COM:UISMENUSINI.COM
```

This will define the necessary symbols and logical names.

To use UISMENUS with LITES2 it may be started before LITES2 as a subprocess using the command:

```
$ SPAWN/NOWAIT UISMENUS menu-file-name
```

or from within LITES2 using the commands

```
* CREATE MAILBOX 1      !  create LSL$LITES2AUX mailbox
* SPAWN/NOWAIT UISMENUS menu-file-name
```

As UISMENUS runs as a separate process, it can outlive LITES2 unless care is taken to ensure that the only route out of LITES2 also stops UISMENUS. UISMENUS will terminate normally as a result of an appropriate menu command (ie return codes 998,997 and 991).

3.5 *Button box*

A Laser-Scan button box (Mapstation console) can be used as an input device. This is connected to a separate serial line, which must have the logical name LSL$CONSOLE. Before the buttons can be used, they must be defined as a PUCK on device 5. This defines the number of buttons that are to be used. The individual buttons can be programmed using MACRO commands as usual.

The buttons are activated by the ENABLE BUTTONS command. If this is given in INITIAL state, and a suitable PUCK command has also been given, then the buttons that are available for use on the button box will light up when the map is read in. When a map has been read in, the availability of the button box can be controlled by the ENABLE, DISABLE and TOGGLE BUTTONS commands.

APPENDIX A

Colour Table

The following is an example of a file describing the colours to be used. It should be set up on logical name LSL$VAX_COLOUR. The character ';' introduces a comment. The colours are specified as proportions of red, green, and blue, in hexadecimal in the range 0-FF. An example file is in LSL$LITES2CMD:VAX.COL

```
; system constants file for VAXstation GPX
; COLOUR DEFINITIONS
;       Red     Green   Blue    Blink   Comment
16                                      ; Number of colours
         0       0       0       0      ; 0
         0       FF      FF      0
        FF       0       0       0
        FF       0      FF       0
         0      FF       0       0
         0      80      FF       0
        FF      97       0       0
        FF      86      FF       0
         0      AF       0       0
         0      BE      FF       0
        FF      C0       0       0
        FF      B3      FF       0
         0      FF       0       0
         0      FF      FF       0
        FF      FF       0       0
        FF      FF      FF       0
```

APPENDIX B

Screen menu

The following is an example of a file that sets up a screen menu. It should be called LSL$LITES2CMD:SCREEN.LCM

```
! SCREEN.LCM
!
! Definition of screen menu
! ========================
!
! first define puck
!
%PUCK 1 32 SCREEN
!
! and describe what SCREEN is to look like
!
%DESCRIBE SCREEN 1 32 6 25.0 200.0 1.0 0.0 screen LSL
!
! now define the contents of each button (box) of SCREEN
!
%MACRO SCREEN2  #%START          #%ENDM
%MACRO SCREEN3  #%START#%END     #%ENDM
%MACRO SCREEN4  #%CURVE          #%ENDM
%MACRO SCREEN5  #%FIND           #%ENDM
%MACRO SCREEN6  #%END            #%ENDM
%MACRO SCREEN7  #%CLOSE          #%ENDM
%MACRO SCREEN8  #%CLOSE SQUARE   #%ENDM
%MACRO SCREEN9  #%MOVE           #%ENDM
%MACRO SCREEN10 #%ROTATE         #%ENDM
%MACRO SCREEN11 #%GET 3          #%ENDM
%MACRO SCREEN12 #%GET 4          #%ENDM
%MACRO SCREEN13 #%INVISIBLE      #%ENDM
%MACRO SCREEN14 #%REPEAT         #%ENDM
%MACRO SCREEN15 #%WINDOW MAP#%PING      #%ENDM
%MACRO SCREEN16 #%ABANDON        #%ENDM
%MACRO SCREEN17 #%SELECT ALL     #%ENDM
%MACRO SCREEN18 #%ZOOM           #%ENDM
%MACRO SCREEN19 #%ZOOM .5        #%ENDM
%MACRO SCREEN20 #%DRAW SCREEN    #%ENDM
%MACRO SCREEN21 #%DRAW MAP       #%ENDM
%MACRO SCREEN22 #%DELETE         #%ENDM
%MACRO SCREEN23 #%EDIT           #%ENDM
%MACRO SCREEN24 #%REMOVE         #%ENDM
%MACRO SCREEN25 #%BRIDGE         #%ENDM
```

```
%MACRO SCREEN26 #%SPLIT         #%ENDM
%MACRO SCREEN27 #%USER 1        #%ENDM
%MACRO SCREEN28 #%USER 2        #%ENDM
!%MACRO SCREEN29        #               #%ENDM
!%MACRO SCREEN30        #               #%ENDM
!%MACRO SCREEN31        #               #%ENDM
%MACRO SCREEN32 #%ABANDON       #%ENDM
!
! now what is to be written in each box
!
%DESCRIBE MACRO SCREEN2 Start
%DESCRIBE MACRO SCREEN3 Symbol
%DESCRIBE MACRO SCREEN4 Curve
%DESCRIBE MACRO SCREEN5 Find
%DESCRIBE MACRO SCREEN6 End
%DESCRIBE MACRO SCREEN7 Close
%DESCRIBE MACRO SCREEN8 Close Squ
%DESCRIBE MACRO SCREEN9 Move
%DESCRIBE MACRO SCREEN10 Rotate
%DESCRIBE MACRO SCREEN11 Get 3
%DESCRIBE MACRO SCREEN12 Get 4
%DESCRIBE MACRO SCREEN13 Invisible
%DESCRIBE MACRO SCREEN14 Repeat
%DESCRIBE MACRO SCREEN15 Window
%DESCRIBE MACRO SCREEN16 Abandon
%DESCRIBE MACRO SCREEN17 Sel all
%DESCRIBE MACRO SCREEN18 Zoom
%DESCRIBE MACRO SCREEN19 Zoom .5
%DESCRIBE MACRO SCREEN20 Draw screen
%DESCRIBE MACRO SCREEN21 Draw map
%DESCRIBE MACRO SCREEN22 Delete
%DESCRIBE MACRO SCREEN23 Edit
%DESCRIBE MACRO SCREEN24 Remove
%DESCRIBE MACRO SCREEN25 Bridge
%DESCRIBE MACRO SCREEN26 Split
%DESCRIBE MACRO SCREEN27 User 1
%DESCRIBE MACRO SCREEN28 User 2
!%DESCRIBE MACRO SCREEN29 MacroA
!%DESCRIBE MACRO SCREEN30 MacroB
!%DESCRIBE MACRO SCREEN32 MacroC
%DESCRIBE MACRO SCREEN32 Abandon
```

APPENDIX C

UISMENUS

MODULE UISMENUS

FUNCTION

UISMENUS is a utility to generate a set of screen menus and allow the user to control programs by probing menu boxes. Prior to execution the user must define a 'tree' of menus. Sequences of menus from the tree can then be displayed on the screen. Probing a menu box either results in a new menu being displayed or a user-defined text being posted to a specified mailbox or a DCL symbol being set.

Used in conjunction with LAMPS utilities that accept mailbox input, UISMENUS allows input from a screen-based set of menus in addition to the normal use of the keyboard.

UISMENUS is restricted to use on VAXstation graphics workstations that are running the VWS (UIS) graphics software.

FORMAT

$ UISMENUS menu-file-name

Command qualifiers	**Default**
/LOGICAL='mailbox-name'	/LOGICAL="LSL$LITES2AUX"
/SYMBOL='symbol-name'	/SYMBOL="LSL$UISMENUSTEXT"

PROMPTS

_List file: menu-file-name

PARAMETERS

menu-file-name

- specifies an ASCII file that contains the names of those files which define the menus to be included in the menu tree. The input name is parsed against LSL$CDL:UISMENUS.DAT;0. This file also allows the user to specify a colour scheme for the menu tree.

COMMAND QUALIFIERS

/LOGICAL
/LOGICAL="LSL$LITES2AUX" **(default)**

- this qualifier is used to specify the logical name that is assigned to the mailbox to which user-specified text is posted. This mailbox is created when UISMENUS is initialised. By agreeing on the name of this mailbox UISMENUS can be used to communicate with a variety of LAMPS utilities. The default logical name allows connection to LITES2.

/SYMBOL
/SYMBOL="LSL$UISMENUSTEXT" **(default)**

- this qualifier is used to specify the symbol name that is to be set to the user specified text. Once this symbol is set UISMENUS will complete and the DCL symbol may then be used in subsequent DCL code.

DESCRIPTION

General

UISMENUS is a LAMPS utility that creates a tree of menus. These menus are composed of buttons (ie boxes) and informational text. When a button is probed one of the following several actions is performed.

- o another menu in the tree is made visible
- o a line of user-defined text is written to a mailbox. This mailbox is created when UISMENUS is first invoked and has a logical name specified by the /LOGICAL qualifier.
- o a DCL symbol is set if the /SYMBOL qualifier has been given.
- o the utility is terminated

To 'probe' a button the screen pointer must be placed on the appropriate part of the menu and the left-hand mouse button pressed. The appropriate part of the menu is termed an active area and is indicated by a change in background colour when the screen pointer is placed in it. If the left-hand button is pressed when the pointer is not in an active region the menu is popped to the front. Pressing the right-hand mouse button when the pointer is on a menu has one of two effects: if the menu is obscured in any way then it is popped to the

front, alternatively if it is not obscured then it is pushed to the back.

Return Codes

Individual menus are defined using a CDL file. These ASCII files provide a description of the menu together with a means of associating a 'return code' and 'return text' with each button in the menu. The action of probing a button depends on the combination of these user-defined quantities, UISMENUS makes special use of the 'return codes' defined by the user:

- Return Code **999**: the return text is interpreted as the name of the menu to be displayed next. All those menus above it in the tree and including the specified menu will be displayed, all other menus that are currently displayed but do not fit this specification are removed from the screen and UISMENUS awaits the next menu event.

- Return Code **998**: the return text is posted to the mailbox specified by the /LOGICAL qualifier and UISMENUS completes once the mailbox has been read.

- Return Code **997**: the return text is set to be the contents of the symbol either specified by the /SYMBOL qualifier or called LSL$UISMENUSTEXT by default, and UISMENUS completes.

- Return Code **996**: all the buttons defined as TOGGLEs within the current menu are set to 'true', and are all forced to appear in inverted foreground and background colours. This does not change the state of buttons within a GROUP of CHOICE buttons. UISMENUS awaits the next menu event.

- Return Code **995**: all the buttons defined as TOGGLEs within the current menu are set to 'false', and are all forced to appear in normal foreground and background colours. This does not change the state of buttons within a GROUP of CHOICE buttons. UISMENUS awaits the next menu event.

- Return Code **994**: the return text is interpreted as the name of the menu to be displayed next in a manner similar to return code 999. In addition all the buttons defined as TOGGLEs within the current menu are set to 'false', and are all forced to appear in the normal colours. Buttons within a GROUP of CHOICE buttons are reset to their initial states. UISMENUS then awaits the next menu event.

- Return Code **993**: posts a text to the mailbox and invokes a specified menu. It combines the functions of return codes 001 and 999 in response to a single button press. The text string to be posted and the menu name are given in one return text separated by a delimiting character that must also be the first and last characters in the text (inside the double quotes). The delimiting character can be any keyboard character other than a double quote or any used in the mailbox text or menu name. UISMENUS then awaits the next menu event.

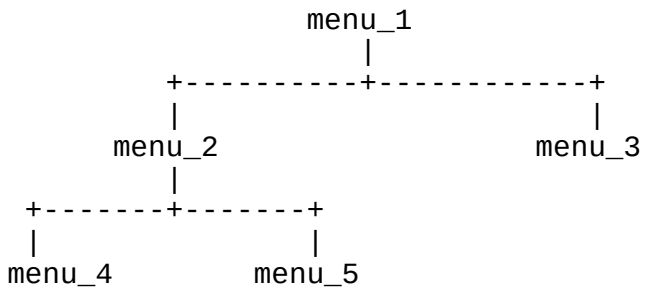
- Return Code **992**: posts a text to the mailbox and invokes a specified menu which is reset. It combines the functions of return codes 001 and 994 in response to a single button press. The text string to be posted and the menu name are given (in this order) in one return text separated by a delimiting character that must also be the first and last characters in the text (inside the double quotes). The delimiting character can be any keyboard character other than a double quote or any used in the mailbox text or menu name. UISMENUS then awaits the next menu event.

- Return Code **991**: UISMENUS completes without doing anything. Note that it cannot complete if any previous mailbox postings are outstanding.

- Return Code **001**: the return text is posted to the mailbox specified by the /LOGICAL qualifier and UISMENUS awaits the next menu event.

Hierarchy

The CDL file that defines an individual menu is indicated by the presence of the command FILE in the menu file, followed by a filename. Any part of the filename not supplied is parsed against the default LSL$CDL:DEFAULT.CDL;0. Legal formats for CDL files are given in the section 'CDL Files'. The ordering of the CDL files in the menu file must be such that the parent of each menu has already been defined before an attempt is made to define one of its children. Thus a menu tree of the form:



could be equally well defined by the orderings: {1,2,3,4,5} or {1,2,4,5,3} or {1,3,2,5,4}. Note that in all cases menu_1 is defined first.

Colours

The menu file contains commands to set the colours of the menus. There are three colours used to define a menu colour scheme:

- FOREGROUND: the colour of the text and linework on the menu,

- BACKGROUND: the colour of the menu on which the text and linework is drawn and,

- HILITE: the colour the background turns to when the pointer is placed on an active area.

A colour command begins with one of the three keywords FOREGROUND, BACKGROUND or HILITE (indicating the colour being specified) followed by an RGB triple expressed as three real numbers separated by white spaces.

EXAMPLE

The following menu file might be used to generate the menu tree shown in the diagram above, the menus would have green text on a black background with an orange high-light. Comments may be included in the file if prefixed by a '!'.

```
!
! E X A M P L E   M E N U   F I L E
!
! Specify the colour scheme for the menu tree
FOREGROUND 0.0 1.0 0.0
BACKGROUND 0.0 0.0 0.0
HILITE     1.0 0.5 0.0
!
! and then the menus themselves...
FILE MENU1
FILE MENU2
FILE MENU3
FILE MENU4
FILE MENU5
!
```

CDL FILES

Individual menus are defined by constructing ASCII files containing Console Definition Language (CDL) commands. A separate file must be constructed for each menu. Currently the following set of CDL commands is supported:

The WINDOW commands must come at the start of the CDL file.

WINDOW POSITION xpos ypos
- specify the position of the bottom left hand corner of the menu in pixels measured from the bottom left hand corner of the screen.

WINDOW SIZE xsize ysize
- specify the size of the menu window in pixels.

WINDOW NOFRAME
- indicate that the menu window is not to have a frame around it. This means that the menu cannot be moved.

The NAME commands are used to define the structure of the menu tree. The menu names so defined are used to establish connections between menus and some applications use them to indicate which menu is to be made visible.

The NAME commands, if they are being used, come after the WINDOW commands but before all the others.

NAME CONSOLE "text"
- give a name to the menu that is being defined. This is required for menus that form a menu tree.

NAME PARENT "text"
- give the name to the menu that is this menu's parent in the menu tree. Note that if this is the first menu in the tree to be defined then the parent name "cdl$none" should be given instead.

The following commands can be used in any combination in any order.

ADD LINE xstart ystart xend yend
- add a line, in the foreground colour, to the menu. The line is drawn from a start position to an end position specified relative to the bottom left hand corner of the menu.

ADD TEXT xoff yoff "text"
- add the specified informational text, in the current font, to the menu. The bottom left hand corner of the text is at the position specified (xoff yoff) relative to the bottom left hand corner of the menu.

BUTTON TEXT xoff yoff "text"
- create a button in the menu, labelled with the specified text. The active area of the button is the area required to print the specified button text in the current font. The bottom left hand corner of the active area is at the position specified (xoff yoff) relative to the bottom left hand corner of the menu.

BUTTON ICON xoff yoff
- create a button in the menu represented by the currently defined icon (see ICON FILE command). The active area of the button is the area required to draw the current icon. The bottom left hand corner of the active area is at the position specified (xoff yoff) relative to the bottom left hand corner of the menu.

CHOICE TEXT xoff yoff "text"
- create a button in the menu, labelled with the specified text. The button forms one of a group of at least two where probing a button will invert the colours (ie turn it on) and force all other buttons in the group to normal colour (ie turn them off). One button in a group of choices must always be on and when the menu is first invoked this will be the first button defined in the group. A series of CHOICE commands must be indicated by a prior GROUP command. The active area of the button is the area required to print the

specified button text in the current font. The bottom left hand corner of the active area is at the position specified (xoff yoff) relative to the bottom left hand corner of the menu.

FONT number
- specify the font to be used to display subsequent informational and button texts. There are twelve fonts available numbered 0 to 11. For example FONT 0 is a font with characters formed in an 8 x 14 pixel cell.

GROUP
- indicate that the subsequent CHOICE buttons are to be considered as a group. Additional groups of choices are indicated by subsequent GROUP commands. A GROUP command must prefix at least one CHOICE commands.

ICON FILE "filename"
- specify the name of the file defining an icon. This file is then accessed by subsequent BUTTON ICON commands to draw an icon in the menu.

ON BORDER
- indicate that the active area of subsequent buttons is to be given a border.

OFF BORDER
- indicate that the active area of subsequent buttons is not to be given a border.

OFFSET xoff yoff
- specify an offset which is applied to all subsequent positions expressed in pixels from the bottom left hand corner of the menu.

RETURN CODE number
- specify the 'return code' for subsequent buttons. The use of this is application-dependent.

RETURN TEXT "text"
- specify the 'return text' for subsequent buttons. The use of this is application-dependent.

TOGGLE TEXT xoff yoff "text"
- create a button in the menu. When the button is pressed the foreground and background colours within the active area are transposed (toggled). The active area of the button is the area required to print the specified button text in the current font. The bottom left hand corner of the active area is at the position specified (xoff yoff) relative to the bottom left hand corner of the menu.

As an example, consider the following CDL file. Comments may be included in the file if prefixed by a '!'.

```
!
! E X A M P L E   C D L   F I L E
!

! menu placed at top-right of screen
window pos 600 600
window size 88 56

! this is 'menu_4' with a parent menu of 'menu_2'
name console "menu_4"
name parent "menu_2"

! text is to be output in font 0, and buttons are not to have a border
font 0
off border

! this button causes 'menu_2' to be displayed
return code 999
return text "menu_2"
button text 0 0  "  E X I T  "

! subsequent buttons post text to the mailbox
return code 1

! this button posts the LITES command 'zoom 10' to the mailbox
offset 0 14
return text "zoom 10"
button text 0 0  " ZOOM LOTS "

! this button posts the LITES command 'zoom 1' to the mailbox
return text "zoom 1"
button text 0 14 "  CENTRE   "

! this button posts the LITES command 'zoom 0.5' to the mailbox
return text "zoom 0.5"
button text 0 28 " ZOOM OUT  "

! this button posts the LITES command 'select all' to the mailbox and
! clears previous selections made by TOGGLE buttons
return code 995
return text "select all"
button text 0 40 " SELECT ALL"

! reset code to post text to mailbox
return code 001

! this button posts the LITES command '%select fc 10' to the mailbox
return text "%select fc 10"
toggle text 0 55 "   ROADS   "

! this button posts the LITES command '%select fc 15' to the mailbox
return text "%select fc 15"
toggle text 0 70 "   RIVERS  "
```

```
! this button posts the LITES command '%select fc 22' to the mailbox
return text "%select fc 22"
toggle text 0 85 "  CONTOURS "

! start a group
group

! this button posts the LITES command '%enable big' to the mailbox
return text "%enable big"
choice text 0 100 " BIG CURSOR "

! this button posts the LITES command '%disable big' to the mailbox
return text "%disable big"
choice text 0 115 "SMALL CURSOR"

! define an icon
icon file "LSL_LOGO"

! this places an icon in the menu
return text ""
button icon 0 130

!
! E N D   O F   E X A M P L E   C D L   F I L E
!
```

The menu is positioned near the top of the screen (although it can subsequently be picked up and moved). The screen is 1024 by 864 pixels.

The names of menus are used to define the menu tree. Note the 'NAME PARENT' command must come after the 'NAME CONSOLE' command and before the button definitions. The menu being defined is menu_4.

The menu is 88 by 56 pixels. Using font zero this translates into a menu of 11 by 4 characters. Each of the four rows is used to represent a different button. The first one tells UISMENUS to display the menu 'MENU_2'. Since 'MENU_2' is the parent of this menu this has the same effect as exiting from it. The other three buttons pass LITES2 commands to the mailbox.

An example tree of menus is contained in the directory

LSL$PUBLIC_ROOT:[LITES2.EXAMPLES.CDL].

This holds a valid set of .DAT, .CDL and .ICON files. To demonstrate this set of example menus they should first be copied to LSL$SITE_ROOT:[LSL.CDL] and LSL$CDL defined to point to this directory by invoking the initialisation file UISMENUSINI.COM with the command:

```
$ @LSL$COM:UISMENUSINI.COM
```

Then the command:

```
$ UISMENUS EXAMPLE.DAT
```

will start these example menus.

ICON FILES

Individual icons are defined by ASCII files containing an encoded description of the bitmap used to display the icon. A separate file must be constructed for each icon. These icon files can be 'manually' constructed using the methods described below. Complex or large numbers of icons may be prepared under consultancy arrangements by Laser-Scan using in-house software.

An icon can be considered as a rectangular block of pixels where the drawn shape is produced by setting the appropriate pixels 'on'. This block is represented in the file by an array of zeros and ones equivalent to the pixels where zeros are background colour and ones are foreground colour. The number of rows and columns in the array should be a multiple of 8. These rows of zeros and ones are encoded into their stored form by converting every 8 digits along a row into the equivalent pair of hexadecimal numbers.

The first line of the file is always a '[' character and the last line is always a ']' character. The second line always specifies the size in the form:
s nrows ncolumns

An icon will be designed on graph paper (Imperial sized divided into eighths of an inch is suggested, as it naturally breaks the icon up into the required 8 pixel units) with shaded in squares representing '1's for the image required.

As an example, consider the following preparation of an icon file to represent an uparrow shape.

This is the graphical design considered as an array of 0's and 1's

```
1111111111111111
0000000110000000
0000000110000000
0000000110000000
1111100110011111
0000100110010000
0000100110010000
0000101111010000
0000110110110000
0001100110011000
0011000110011100
0110000110000110
1100000110000011
1000000110000001
0000000110000000
1111111111111111
```

This is the array encoded row for row in hexadecimal form.

```
1111111111111111 =   ff ff
0000000110000000 =    1 80
                     81 81
     etc             c1 83
                     61 86
                     31 8c
                     19 98
                      d b0
                      b d0
                      9 90
                      9 90
                     f9 9f
                      1 80
                      1 80
                      1 80
                     ff ff
```

This is the final stored form of the encoded array and is a listing of the file LSL$PUBLIC_ROOT:[LITES2.EXAMPLES.CDL]UP1ARROW.ICON

```
[
s 16 16
  ff  ff   1  80  81  81  c1  83  61  86  31  8c  19  98   d  b0
   b  d0   9  90   9  90  f9  9f   1  80   1  80   1  80  ff  ff
]
```

MESSAGES (ERROR)

These messages indicate an error in processing which will cause the program to terminate. The most likely causes are a corrupt or otherwise invalid input file, or an error related to command line processing and file manipulation.

BADCOLOUR, error specifying menu colour

Explanation: the menu file contained a line with either the FOREGROUND, BACKGROUND or HILITE command. There colour specification following the command was in error.

User action: edit the menu file to ensure that the colour specification is correct.

BADINPUT, error with input on line 'integer' of menu input file 'filename'

Explanation: there was some unexpected form of input on the specified line of the menu input file.

User action: edit the menu input file to ensure that is legal.

CLSFIL, error closing menu input file 'filename'

Explanation: there was an error when attempting to close the specified menu file.

User action:

ERRINPUT, error reading command from menu input file

Explanation: unexpected error while reading from menu input file.

User action: try to type out the menu file at a terminal and look for errors.

FAILCREATE, fail to create a mailbox with name "'logical-name'"

Explanation: UISMENUS is unable to create a mailbox (with the quoted logical name).

User action: ensure that the quoted logical name is legal.

FAILMBX, fail to find a mailbox with name "'logical-name'"

Explanation: UISMENUS is unable to find the mailbox (with the quoted logical name).

User action: ensure that the quoted logical name is legal.

FAILMENU, fail to create menu from file "'filename'"

Explanation: UISMENUS was unable to create a menu from the specified CDL file.

User action: ensure that the specified CDL file is legal. In particular check for the presence of the 'name console' and 'name parent' commands.

NOCDLFILE, no CDL filename specified

Explanation: the menu file contained a line with the FILE command but no filename.

User action: either remove the empty FILE command line from the menu file or complete the command by inserting a CDL filename.

OPNFIL, error opening menu input file 'filename'

Explanation: there was an error when attempting to open the specified menu file.

User action: type the menu file at the terminal to ensure it is valid.

UNKPRIM, unexpected command 'command'

Explanation: an unexpected primary command was found in the menu file; legal primary commands are FILE, FOREGROUND, BACKGROUND and HILITE. In addition comments lines are allowed if prefixed by a '!'.

User action: edit the menu file to ensure that it only contains legal primary commands.

MESSAGES (FATAL)

These messages indicate a severe error in processing, or some form of system failure, which has caused the program to terminate.

ABORT, previous errors invalidate run - UISMENUS aborting

> **Explanation:** UISMENUS has failed as indicated by a previous error. There is no point in continuing so UISMENUS will complete.
>
> **User action:** Fix the problem(s) that gave rise to the earlier error messages.

MESSAGES (OTHER)

In addition to the above messages which are generated by the program itself, other messages may be produced by the command line interpreter (CLI) and by Laser-Scan libraries. In particular, messages may be generated by the Laser-Scan I/O library, LSLLIB. LSLLIB messages are introduced by '%LSLLIB' and are generally self-explanatory. They are used to explain the details of program generated errors.