Laser-Scan Ltd.

LITES2

Programmer's Guide

Release 0.3 - 28-Apr-1987

Copyright (C) 2019 Laser-Scan Ltd
Science Park, Milton Road, Cambridge, England CB4 4FY tel: (0223) 420414

Document "LITES2 Programmer"		Category "TECHNICAL"
Document Issue 0.0	Clarke Brunt	19-Oct-1984
Document Issue 0.1	Paul Hardy	09-Jan-1985
Document Issue 0.2	Clarke Brunt	21-Jan-1985
Document Issue 0.3	Ron Russell	28-Apr-1987

CONTENTS

1 *General Structure and Conventions*

This space intentionally left blank.

2 *Program Structure*

This space intentionally left blank.

3 *Command Handling*

This space intentionally left blank.

4 *Data Structures*

LITES2 maintains, in memory, a data structure known as the *sector list*, which permits rapid access to features close to any particular point on the map. This structure is used by the 'find' type command (FIND, RECOVER, SEARCH), and for window drawing, so that a search of the entire IFF file(s) is not necessary.

The working area, bounded by array LIMITS in COMMON /MASTER/, is divided into a number of rectangular *sectors* (user settable, up to a preset maximum of 200). Sectors are numbered starting at 1 in the bottom left corner, increasing across then up, An entry is made in the sector list each time a feature enters a sector. For line type features, there will be several entries for the same sector if the feature reenters it several times.

The sector list is organised in such a way that rapid access is possible to all entries in a given sector (these are more or less contiguous in memory), and to all the entries for a given feature (by pointers).

The entry in the sector list is known as a *pack*. The sector list itself is accessed by a number of low level routines, which transfer packs between an internal packed format (20 bytes), and an array of 10 longwords (40 bytes) used by higher level routines. The layout of the longword array is defined in PARAMETER /PACK/ which sets parameters for the length of the array (PAKLEN) and the array subscripts within it. These are:

PAKLNK - Pointer (absolute address) to previous pack of feature (circular)
PAKPNF - IFF pointer to start of feature (NF or leading TC, CH, or VO)
PAKPST - IFF pointer to implicated ST (lines only)
PAKFC - Feature Code
PAKSEG - GKS Segment number
PAKVRT - First implicated vertex within ST, starting at 1 (lines only)
PAKOVL - Layer index, starting at 0
PAKGT - Graphical Type
PAKMAP - Map number (IFF file) starting at 0
PAKFLG - Flags

The bit positions within the PAKFLG longword (masks) are:

DELFLG - This pack is deleted (marked available for re-use)
FSTFLG - This pack is the first for this feature
LSTFLG - This pack is the last for this feature
DRAFLG - This feature has been drawn (in window redraw)
FINFLG - This feature has been found (in spiral search)
LIMFLG - This feature is in limbo state

The internal packed data is stored in virtual memory obtained by a call to SYS$EXPREG in function GETBLK. Variables controlling the memory allocation are in COMMON/SECMAP/. Routines PUTPAK and GETPAK (which are coded in macro - MAR:PACKER.MAR), move data between the longword array and the internal packed array, the layout of which is defined by these routines and is liable to change. Routines at the next higher level are:

LOGICAL FUNCTION ADDPAK(PACK, SECTOR)
Add the data from the longword array PACK to the chain for SECTOR, replacing a deleted pack if possible, setting the PAKLNK field of the pack to point to the previous pack for this feature. Returns .FALSE. if successful, .TRUE. if fails

which can only happen if more virtual memory was needed but could not be obtained by GETBLK, in which case a system error message should have been printed. The chaining of packs for a given feature is controlled by the settings of the flags in PACK(PAKFLG). The first pack for a feature must have FSTFLG set, the last must have LSTFLG set. Debugging code in ADDPAK checks that these are used correctly.

LOGICAL FUNCTION SETPAK(SECTOR)
Set up pointers in COMMON/SECMAP/ so that a call to NXTPAK (see below) will get the first pack for SECTOR. Returns .FALSE. if successful, .TRUE. if SECTOR is empty.

LOGICAL FUNCTION NXTPAK(PACK)
Read the next pack for the current sector into longword array PACK. Use SETPAK (see above) before first call. Returns .FALSE. if successful, .TRUE. if there are no more packs on the chain. Deleted packs are ignored completely.

SUBROUTINE SCNFTR(LINK, FUN)
Scan a complete feature chain of packs, calling routine FUN(PACK,PLINK), where PLINK is the address of the pack, for each pack. Suitable routines are DELPAK and CHAPAK.

SUBROUTINE DELPAK(PACK, LINK)
Mark PACK deleted (available for re-use), then replace it in location LINK. The absolute address LINK will probably be obtained from the PAKLNK field of another pack. Only complete feature chains of packs should be deleted, hence this routine should only be used as an argument to SCNFTR, as in CALL SCNFTR(LINK,DELPAK).

SUBROUTINE CHAPAK(PACK, LINK) Make various changes to the contents of a pack, then replace it into location LINK. Used as an argument to SCNFTR, as in CALL SCNFTR(LINK,CHAPAK).

SUBROUTINE SCNSEC(SECTOR, FUN)
Uses SETPAK and NXTPAK to scan all packs for the given SECTOR, optionally testing each with function RELVNT, and then applying logical function FUN(PACK) to each. FUN must return .TRUE. if scanning is to continue.

LOGICAL FUNCTION RELVNT(PACK)
Returns true if the pack is currently selected. Tests the map, layer, FC, limbo state, and two specific features to ignore.

SUBROUTINE CLRFLG(MASK, PACK)
Clear bits in the flags entry of an *internal* pack.

SUBROUTINE GETFLG(FLAG, PACK)
Reads flags from an *internal* pack into longword FLAG.

SUBROUTINE SETFTR(MASK, LINK)
Sets bits in the flag entries in all packs for a feature. LINK is the address of one of the packs.

SUBROUTINE CLRSEC(MASK, SEC)
Clears bits in the flag entries in all packs for a sector.

5 *Handling of IFF files*

LITES2 can simultaneously handle up to MAXMAP (currently 9) IFF files. COMMON /MAPS/ contains the details of the maps. Since the IFF library can only handle a lesser number of simultaneously open files, the files must be opened and closed a required. N.B. Internally (as stored in packs), the map number starts at 0, whereas the user treats the first map as 1.

Routine SELIF(MAP) selects the given IFF file using IFFSEL. If the file is not open, then OPENIF is called. Routine OPENIF(MAP(will open the given IFF file removing trailing EO,EM,EJ entries, using a free IFF LUN if possible, or using CLOSIF to close the file open on the 'least recently selected' LUN. Routine CLOSIF(MAP) closes the given IFF file, terminating it with EO,EM,EJ if needed.

6 *Initial reading of IFF files*

Routine INPUT performs initialisation and reading of IFF files. Each file is read by routine IFFBDY, which copies entries from the .IFF file to the .WRK file if required. The details of each feature are put into variables called COI... or OI... (current output item) in COMMON /OIIH/, including 'static' parts of the pack COIPAK (NF pointer, FC, layer, GT, map). For each ST entry, routine FLUSH is called, which is responsible for filling in the remaining details of COIPAK (ST pointer, vertex number) and using ADDPAK to add packs for all implicated sectors. FLUSH optionally calls STDRAW to draw the feature, and adds ST, RO, TX, EF entries as required. For circle arcs, symbols, and texts IFFBDY uses GENAUX to generate an auxiliary array (real - layout defined by PARAMETER /AUXDEF/) to hold geometric details of the feature in a consistent format. The auxiliary array, used by both FLUSH and STDRAW is not stored, but is regenerated each time a feature is referenced.

7 *Graphics and Drawing*

7.1 *GKS Interface*

Graphics are optional in LITES2 - the program will function in an identical fashion whether or not graphics are enabled. LITES2 uses GKS output primitives polyline (GPL) and fill area (GFA) plus the GKS routines to set attributes for these. LITES2 represents each feature by a unique GKS segment. On a full GKS system (e.g. Sigmex 6100 series workstation), a segment may be deleted, set visible/invisible, or highlighted by reference to its segment name. When using a minimal GKS look-alike (e.g. GKSLDLIB for TEK 4014 and Sigma ARGS), these functions must be simulated, so LITES2 calls the following routines to perform segment functions. ('L' is substituted for GKS 'G' in names.) These routines are implementation dependent and may either just call the equivalent GKS function or simulate it.

LCRSG(PACK) Create segment. GCRSG(PACK(PAKSEG))
Merely note the name of the currently open segment, and the fact that it is visible.

LCLSG(PACK) Close segment. GCLSG(PACK(PAKSEG))
Reset displays to default state and note that no segment is open.

LSVIS(PACK, IVIS) Set segment visibility. GSVIS(PACK(PAKSEG), IVIS)
Features put into limbo are made invisible. They are drawn in black on the ARGS, and are scrubbed on the TEK. This routine is called when deleting, recovering, or creating deleted features (as in PART DELETE).

LDSG(PACK, VISI) Delete segment. GDSG(PACK(PAKSEG))
Segments are deleted when a feature is completely removed. This occurs when a feature is edited and the old version discarded (VISI .TRUE.), or when limbo features are finally removed (VISI .FALSE). (The latter never happens at present.) When a segment a deleted, it is drawn in black on the ARGS, but is left alone on the TEK (*not* scrubbed).

7.2 *Drawing*

A complete feature is drawn by a call to routine DRAFTR(PACK). This uses routine STDRAW (store draw) to draw each ST entry of the feature. For features being read in, or created as a result of editing, DRAFTR is not used - instead FLUSH calls STDRAW directly, in addition to calling the GKS segment 'housekeeping' routines.

7.3 *Window Redrawing*

Routine FLVIEW (file view) performs redrawing operations. On an intelligent GKS workstation, this consists of little more than 'redraw all segments on workstation', but in the case of a non-intelligent workstation, the required features are redrawn from the IFF file. FLVIEW determines the sectors which lie at least part on the screen, and clears the DRAFLG bit in all their packs by calls to CLRSEC. It then calls SCNSEC for each in turn with DRAWDP (draw data pack) as function. DRAWDP sets the DRAFLG bit for all features inspected, and

for those not already drawn, calls DRAFTR to actually draw the feature.

8 *Finding and Searching*

8.1 *Finding*

The FIND operation entails a scan of up to four sectors around the current cursor position, looking for up to four possible struck items. Routine GROPE determines the sectors to be scanned, then calls SCNSEC for each of them, with FIND as the function to apply to each pack. FIND enters details of the best 4 items into the OBJ... arrays in common /STRIKE/, and GROPE then orders them according to distance from the cursor.

8.2 *Searching*

The SEARCH operation scans all sectors, moving out from the cursor position in a spiral pattern. GROPE initialises the search, clearing the FINFLG bit in all packs by calls to CLRSEC, then calling SCNSEC with FIND as function, for each sector. FIND sets the FINFLG bit for each feature it examines, using SETFTR, so that the same feature is not examined several times (once for each of its sectors). FIND stops the scanning as soon as something fitting the criteria in common /CNSTRN/ is found, but the context of the search is preserved by GROPE so that it may be resumed later.

8.3 *Found object*

Having obtained details of one or more struck items in the OBJ... arrays, NXTITM is called to make one of them the found object. Repeated calls to NXTITM will cycle round the struck items, provided that there is more than one. The main work of making a feature the found object is performed by HAULIN, which transfers the details in the OBJ... arrays to FO... variables, also in /STRIKE/. In addition, HAULIN obtains further details from the IFF file, including the NF number, the presence or absence of TCs and ACs, the first and last coordinates of the feature, the IFF address of the end of the feature. Up to 200 vertices of the feature, centred on the cursor position, are loaded into array FOXY, for use in drawing the refresh picture. HAULIN is also responsible for adjusting the vertices in FOXY, if we move along the feature.

9 *Editing*

9.1 *General*

Any changes to the geometry of a feature (i.e. changing the points of a line, changing the characters in a text) require that the sectors and packs be re-calculated. At present, changes of this type involve the creation of a new feature in the IFF file, and the deletion of the old version. It would be possible to perform some edits of this type in-situ, but this has not been done as yet. The only edits which are performed in-situ are alteration of the FSN, alteration of the FC of lines, and edits to AC's and TC's when the edited entries will fit in the old space.

Editing operations requiring a single feature and no cursor positioning (e.g. DELETE, CHANGE FC, REMOVE, REVERSE) operate on the found object and are completed immediately. Other editing operations (e.g. INSERT, JOIN) requiring two features or cursor positioning cause routine HANDLE to be called. This transfers the contents of the found object (FO...) variables to the object in hand (IH...) variables, thus making the found object the object in hand, and allowing another found object to be identified, possible merely to locate the cursor. This type of edit is terminated by an END command.

9.2 *Example Of Simple Single Feature Edit - REMOVE*

After ensuring that the cursor is actually positioned on a vertex of a line feature, DOREMO copies the found object pack (FOPAK) into the current output item pack (COIPAK), using CPYPAK. The flag entry of COIPAK is set to its default state of 0 (new feature not to be in limbo). The segment number of the found object is to be re-used, otherwise we should have to set COIPAK(PAKSEG) to 0 to force generation of a new segment number. STRTFT (start feature) is called, with FO... variables for all its arguments. This creates the header for the new feature in the IFF file, keeping the same TC's, FSN, FS entry, and AC's as the found object, and also initialises FLUSH. Provided that this succeeds (can only fail if map is read-only), LDSG is called to delete the old segment (and make the segment number available for re-use). ADDVRT is called to add vertices from the found object into the new feature, omitting the one we are removing, the ENDFT is called to flush out and terminate the feature. Finally DELFT is called to void the old feature in the IFF file, and to remove its packs from the sector list. ABANDN to called at the end of all successful editing operations to reset the program to the default READY state.

9.3 *Example Of Two Feature Edit - JOIN*

The same principles apply as in the single feature edit. DOJOIN ensures that the cursor is located on an end vertex of a line feature, then calls HANDLE to make the found object the object in hand, enters EDIT state, and sets NDMODE so that END will despatch to the code to end a join operation.

Since we are now performing a JOIN operation, FIND will only find the ends of line features, and NXTITM automatically positions the cursor midway between the points we are joining.

DOEND despatches to NDJOIN (end join), which after checking that we actually have a found object to join to uses STRTFT to start off a copy of the object in hand. The vertices and cursor position are added into the new feature in such a way as to maintain the direction of digitisation of the object in hand, reversing the found object if necessary. There are of course two features to be removed from the screen and deleted in this case.

9.4 *Example Of Part Feature Edit - MODIFY PART*

After ensuring that the cursor is located on a line feature, the position on the found object is 'marked' (FO... copied into FM...), the cursor is constrained to move along the found object (FNDMDE=1,ROLING=.TRUE.) and ON state is entered.

While moving along the found object, various CHANGE commands may be given. The changes to be made are saved in COMMON /VARIAT/.

When the END command is given, the found object is taken into hand, and CHPART (change part) called to make the changes. CHPART ensures that the marked position comes before the current position in vertex terms along the feature, swapping them if needed. It then creates up to three new features, making the required changes to the middle piece, re-using the FSN and segment number of the old feature if possible, otherwise generating new ones.

9.5 *Example Of Text Or Symbol Edit - ROTATE TO*

All text and symbol editing consists of changes to feature details in COI... variables, which may have been copied there from the found object, or be the result of constructing a new feature. DOROTA sets the new angle and enters MODIFY state, so that the effect of the rotation may be verified before the operation is ended. When the END command is given, DOEND despatches to NDSTCN (end symbol/text construction) which creates the new feature and deletes the old one.

10 *Construction of new line features*

DOSTAR calls STRTER which checks that the specified attributes of the new feature are valid, and then buffers up the first point in STRTXY. Further calls to STRTER buffer up additional points until the buffer is full, when STRTFT is called to start off the new feature, and part of the STRTXY buffer is added into the feature, keeping a part for the refresh picture and for possible removal of points. The feature is finally ended by further calls to STRTER from DOEND.

11 *Editing of AC, TC, and CH entries*

The AC, TC, and CH entries for a particular feature may be loaded into memory by a call to routine GETAC. This is called from NXTITM if VERIFY AC is switched on, and also by the EXAMINE AC, and ANCILLARY commands. The entries are stored in memory forming part of the sector list, with a dummy sector (the AC sector) one

greater than the maximum real sector number. One of the entries in the AC sector is loaded into a FORTRAN array ACBUF, whose layout is defined in COMMON /ACS/. The entries are chained (TC's and CH's first, AC's second) to provide quick access to the first, last, previous, and next entries in the list.

Routine ADDAC adds a new entry to the list, DELAC deletes one. ACEDT replaces the edited entries into the feature using COPYAC, which takes entries from the AC sector if possible, otherwise directly from the IFF file. Routines NEXTAC and PREVAC make the next or previous entry on the list current, TYPAC types the current entry. VERFAC uses these to type the entire list.

12 *Odds and Ends*

This space intentionally left blank.

13 *Device-dependent Interface*

13.1 *LITES2 For Different Hardware*

The LITES2 program is adapted to run on various hardware configurations by linking the main body of the program (OBJ:LITES2.OLB) with a set of device dependent routines (e.g. OBJ:TVES.OLB). The sources for these routines are kept in a separate directory (e.g. [LITES2.TVES]). At present, there are 23 device dependent routines which are called by LITES2, and hence must be supplied. In some versions, these in turn call other routines, so there may be more than 21 routines in the device dependent directory. The device dependent routines may call back routines in the main body of LITES2, and in the case when these routines are not otherwise used, they are explicitly included in the LINK instructions, since LITES2.OLB is scanned before the device dependent library.

13.2 *Specification Of Routines*

The routines are described here grouped by function.

13.2.1 *Initialisation* -

The following routines are called when the final IFF file is specified, and initial read in and draw commences.

```
LOGICAL FUNCTION TSTWK(NUMBER,WTYPE)
arguments
INTEGER NUMBER (input)
INTEGER WTYPE (output)
```

NUMBER is either 1 or 2 to indicate primary or secondary workstation. The function should return .TRUE. if the specified workstation is available in this implementation, in which case WTYPE should be set to the workstation type to be used in the GKS Open Workstation (GOPWK) call.

```
LOGICAL FUNCTION TSTSEG
```

Should return .TRUE. if GKS segments are to be stored on the workstation(s). The variable USESEG in common /WORKS/ is set to the value returned by this function. This is used by FLVIEW in re-draw operations to determine whether features need to be re-drawn from the IFF file, and may also be used in other device dependent routines. TSTSEG should only return .TRUE. if the workstation(s) are actually capable of storing segments, and the variable SEGOPT in common /OPTION/ should be taken into account if the ENABLE SEGMENTS command is to have any effect.

```
LOGICAL FUNCTION TABINIT
LOGICAL FUNCTION BITINIT
LOGICAL FUNCTION BALINIT(ON)
LOGICAL FUNCTION SCRINIT(ON)
LOGICAL FUNCTION BUTINIT(ON)
```

Should initialise the system for reading the digitising table, bitpad, trackerball, screen menu and function butons respectively, returning .TRUE. if this succeeds. In the case of BALINIT, SCRINIT and BUTINIT the argument ON is .TRUE. if the trackerball etc is to be switched on, .FALSE. if it is to be switched off, and the function return indicates the new state (.TRUE. for on). See existing versions (e.g. in [LITES2.TVES]) for examples of what may be necessary.

```
SUBROUTINE INIWK(WKID)
argument:
INTEGER WKID (input)
```

WKID is set to 1 or 2 for primary and secondary workstations. This routine is called immediately after GKS Open Workstation (GOPWK) and should perform any device dependent initialisation of the workstation (behind the back of GKS!). This routine is often null, but provides a hook on which to hang any operations of this type.

13.2.2 *Termination* -

The following routine is called as part of final program rundown for exit.

```
SUBROUTINE CLSTAB
```

This routine should close down the digitising table reading mechanism.

13.2.3 *Digitising Table (and Bitpad) Handling* -

The following routine is called to read a single point from the digitising table or bitpad, when setting up maps, menus, and tracking areas.

```
SUBROUTINE GTDGXY(LINE,BUTTON,XY,EXSTSU,ABORT)
arguments: (all output)
INTEGER*4 LINE    2 for bitpad, 3 for table
INTEGER*4 BUTTON  button pressed (starting at zero)
REAL      XY(2)   coordinate
LOGICAL   EXSTSU  'preserve existing setup' button pressed
LOGICAL   ABORT   'abort setup' button pressed
```

See existing versions for examples of this routine. The routine should probably set ABORT and return if CTRL/C is pressed to enable a way out if the table stops working.

13.2.4 *Refresh Picture Drawing* -

The following routines are concerned with the drawing of the refresh picture on the workstations.

SUBROUTINE RFDRAW

This routine is called from the LITES2 main loop between every primitive command, and should ensure that the refresh picture on the workstation reflects the current state of the program. The actions necessary vary a great deal with different hardware. The components of the refresh picture for which RFDRAW is responsible are:

1. The cursor, plus its position, size, and blinking.
2. The found object.
3. The object in hand.
4. The current construction.
5. Any symbol or text being modified.
6. Rubber band lines.

SUBROUTINE STDINI

This routine is called as the first action in STDRAW if REFNG is false. Its purpose is to prepare the workstations for drawing features in stored mode (i.e. not refresh), such as removing the refresh picture. This routine need only do anything if (a) RFDRAW uses STDRAW, and (b) There is some undesirable interaction between the stored and refresh pictures. It is the responsibility of RFDRAW to set REFNG .TRUE. to suppress the call to STDINI in STDRAW, and to set it back to .FALSE. again.

13.2.5 *Interactive Input* -

The following routine is called by the LITES2 command decoder to obtain input from the interactive controls.

SUBROUTINE INTERACT

In simple terms, this routine must obtain the next line of interactive input, and place it in the buffer TXTBUF. In practice, the routine tends to be rather complex because of the multiplicity of input devices. Routine TRNTIP (translate table input) is provided in the main LITES2 library to interpret input from the table, bitpad, and trackerball, returning a string with appropriate menu or button commands, and a position. See existing versions of INTERACT for examples.

13.2.6 *Graphics Routines* -

The following routines are called at the level of GKS by LITES2, to enable the implementor either to use a real GKS package, or to simulate the effect that would be obtained.

SUBROUTINE LCRSG(PACK) Create segment.
Argument:
INTEGER*4 PACK(PAKLEN) (input) LITES2 pack
GKS function: GCRSG(PACK(PAKSEG))

This routine may either call GCRSG to create a GKS segment, or if segments are not being used should just perform whatever housekeeping is needed to note that a visible segment is now open.

SUBROUTINE LCLSG(PACK) Close segment.
Argument:
INTEGER*4 PACK(PAKLEN) (input) LITES2 pack
GKS function: GCLSG(PACK(PAKSEG))

Either call GCLSG or simulate the closing of a segment.

SUBROUTINE LSVIS(PACK,IVIS) Set segment visibility.
Arguments:
INTEGER*4 PACK(PAKLEN) (input) LITES2 pack
INTEGER IVIS (input) GKS visibility flag (GINVIS,GVISI)
GKS function: GSVIS(PACK(PAKSEG), IVIS)

Called when features are put into limbo (DELETE), or are recovered, or when pre-deleted features are created (PART DELETE, BRIDGE). The routine may either call GSVIS, or simulate its effect e.g. by drawing over in black, or 'scrubbing', to make invisible, or by redrawing normally to make visible.

SUBROUTINE LDSG(PACK,VISI) Delete segment.
Arguments:
INTEGER*4 PACK(PAKLEN) (input) LITES2 pack
LOGICAL VISI (input) .TRUE. if segment is currently visible
GKS function: GDSG(PACK(PAKSEG))

Segments are deleted when a feature is completely removed. This occurs when a feature is edited and the old version discarded (VISI .TRUE.), or when limbo features are finally removed (VISI .FALSE). (The latter never happens at present.) The routine may either call GDSG, or may simulate its effect e.g. by drawing over in black, or even by doing nothing if using a storage display.

SUBROUTINE LSALLI Set all segments invisible
SUBROUTINE LSALLV Set all segments visible

These routines are called by FLVIEW if using segments when selections are in use. They may take advantage of the special facilities usually provided on GKS workstations to perform these operations. If a pure GKS was being used, then it would be necessary to set the segments individually.

SUBROUTINE LRSGWK(WKID) Redraw all segments on workstation
Argument:
INTEGER WKID (input) 1 or 2 for primary and secondary workstation
GKS function: GRSGWK(WKID)

Called by FLVIEW if segments are in use. It is unlikely that this routine will ever be anything other than a call to GRSGWK, plus any associated adjustment of workstation settings.

13.2.7 *Miscellaneous Routines* -

SUBROUTINE HARCOP

This routine should produce a hardcopy if possible. If not, it should call IGNCMD to inform the user that the command does not work.

SUBROUTINE PING
SUBROUTINE RASP

These routines (which are usually both in PING.SRC) should produce noises at the workstation. PING is called to prompt the user (e.g. to digitise map corner points) and at certain other times. It may, for instance, just transmit a bell. RASP is called when MOAN or NASTY error messages are produced. Some versions use two bells, but if the workstation is capable of any other noise (e.g. TKRASP on the MUART TEK4014) then this may be used.

13.2.8 *Utility Routines* -

These routines are not called directly by LITES2, but are often used by other device dependent routines.

SUBROUTINE BOXTXT Used by RFDRAW to refresh text with a box.
SUBROUTINE RFBAND(WKID) Used by RFDRAW to draw rubber band lines.
SUBROUTINE DIGTRN(...) Used by GTDGXY and INTERACT to decode table string.

14 *Licensing system*

14.1 *Overview*

The licensing system within LITES2 depends on a shared image which is built for each individual customer. This is located by LITES2 in the routine INIT through the logical name LSL$LITES2LOCK.

14.2 *The Function LOCKFU*

The shared image consists of one function (called LOCKFU) which is called by LITES2 in INIT and whenever a feature is licensed. It can be called in 4 ways and returns one of:

1. the functions that are licensed
2. the identification numbers of the CPUs that are licensed (note that only the type of the SID is checked for all but VAX780s)
3. the number of users allowed (not used at present)
4. the expiry date of the licence

(2, 3 and 4 also return an announcement string)

The .src file for LOCKFU has an include statement, which includes a file COM:CUSTDAT.SRC. This latter file has data statements that specify the facilities that are allowed at the particular site.

The file CUSTDAT.SRC is built by a program INSTGEN, which prompts for information required by the licence. To create a demonstration licence give a CPU number of -1, when LITES2 will run on any CPU, and all facilities will be available. Having built a suitable parameter file, the function LOCKFU can be compiled and linked into a shared image.

There is a command file that can be used to organise this whole rather complicated procedure.

14.3 *Building The Licensing Function*

To build a licence for a particular customer,

1. first of all set up the LITES2 environment by giving the command @LSL$COM:ILITES2
2. get into the appropriate directory with the command SD COM
3. build the licensing function by giving the command @LICENCE.

NOTE

This command file can take an argument which is the name of the file being created; if this is not given it will be asked for, and the licence will be created with this name (in the directory EXE: with the extension .LIC).
The command file runs the program INSTGEN, which asks for the facilities that are to be licensed. If there is a file with the name given above and the extension .FAC in the directory COM:, then the data will be read from this file, otherwise it will be taken from the terminal.

14.4 *Building A Demonstration Licence*

To build a demonstration licence, use the procedure given above, and give a CPU number of -1.

NOTE

If producing a demonstration licence, always give an expiry limit on the licence.

INDEX