Laser-Scan Ltd.

Feature Representation Library FRTLIB

Programmer Reference Manual

Issue 1.19

Science Park, Milton Road, Cambridge, England CB4 4FY tel: +44 (0) 1223 420414

Document "FRTLIB Reference"   Category "REFERENCE"

Document Issue 0.1	Paul Hardy	14-Jun-1983
Document Issue 0.9	Paul Hardy	29-Oct-1984
Document Issue 1.0	Tony J. Ibbs	10-Dec-1984
Document Issue 1.1	Clarke Brunt	6-Nov-1985
Document Issue 1.2	Ron Russell	1-Jul-1986
Document Issue 1.3	Clarke Brunt	23-Nov-1986
Document Issue 1.4	R J Hulme	06-Jan-1987
Document Issue 1.5	T J Hartnall	29-Jan-1987
Document Issue 1.6	R J Hulme	09-Feb-1987
Document Issue 1.7	R W Russell	20-Oct-1987
Document Issue 1.8	R W Russell	14-Jan-1988
Document Issue 1.9	R W Russell	17-Jun-1988
Document Issue 1.10	R W Russell	16-Nov-1988
Document Issue 1.11	R W Russell	13-Jun-1989
Document Issue 1.12	Clarke Brunt	27-Jul-1989
Document Issue 1.13	Clarke Brunt	25-Jan-1990
Document Issue 1.14	Clarke Brunt	8-Mar-1991
Document Issue 1.15	Ron Russell	5-Mar-1992
Document Issue 1.16	Clarke Brunt	5-Nov-1992
Document Issue 1.17	Clarke Brunt	13-Sep-1993
Document Issue 1.18	Clarke Brunt	30-Aug-1994
Document Issue 1.19	Clarke Brunt	9-Jan-1995

1 *INTRODUCTION*

The Legenda library system has been used in existing LSL graphics programs (SOL, LITES, and SPM) for some years, but it has limitations arising from its origins on PDP11 HRD-1 systems. Recent developments have highlighted requirements for an enhanced system for high quality cartographic work, and this has resulted in the development of a new library (FRTLIB) for manipulating feature representations.

The main differences between LEGLIB and FRTLIB include :-

- Legenda files are single binary files needing a special program to manipulate them (CTG). FRTLIB uses a text file, which can be edited using a standard text editor, together with one or two IFF files to contain the coordinate parts of symbol and text representation. These IFF files can be manipulated using standard LSL IFF utilities (eg LITES).

- LEGLIB goes through the same lookup mechanism for all feature types. FRTLIB can get the attributes of non-patterned lines in a single lookup. Symbols, and patterned lines may require a second reference, but these make up a small part of most maps.

- Symbols in Legendas can only consist of a small number of straight lines. FRT symbols may contain combinations of lines, curves, and circle arcs of almost arbitrary complexity.

- The FRT library supports a much wider range of attributes of a feature in terms of line widths, colours, and sizes.

- LEGLIB has no representation of character fonts, and hence these previously have had to be statically defined. FRTLIB uses the same facilities as for symbols to allow text characters to be defined and manipulated.

- In LEGLIB, the feature code for a symbol was looked up directly in the legenda file. This means that if several feature codes have the same symbol, then the symbol shape must be defined several times. FRTLIB has two tier lookup, and many feature codes may reference the same symbol shape.

- FRTLIB includes routines to plot symbols, texts, and patterned lines, rather than this being the responsibility of individual programs. This ensures consistency between different programs which use the library.

2 *LIBRARY*

FRTLIB allows a feature code (FC) to be looked up in a file or set of files to obtain the correct graphical representation of a line, curve, patterned line, symbol or text feature. The mechanism involves tables of feature representations, together with files of symbol and text character definitions.

For details of the content and structure of the lookup files used by FRTLIB see the FRT User Guide.

FRTLIB is a library of FORTRAN callable subroutines which communicate with the user program via subroutine arguments, function return values, and a series of common blocks. The library is LSL$LIBRARY:LSLFRTLIB and should be scanned before LSLLIB and IFFLIB which it references.

The two latter libraries are also available in shareable image form. To link with the shareable images, specify LSL$LIBRARY:LSLSHR/OPT and LSL$LIBRARY:IFFSHR/OPT on the link command lines.

FRTLIB itself is also available as a shareable image. To link with this, specify LSL$LIBRARY:FRTSHR/OPT on the link command line. If using this, then you must also use the LSLSHR and IFFSHR images. The FRTLIB shareable image cannot call graphics routines directly, since the particular routines required are not fixed. Instead, it calls 19 user-supplied routines (see below) whose addresses must be loaded into an array in common block SRIVEC. The object library, LSLFRTLIB.OLB, contains standard versions of these routines (which call real GKS routines), and also a routine FRT_GRAPH_INIT to load the addresses into the common block. This means that to use graphics with the FRTLIB shareable image, you must include LSLFRTLIB.OLB in the link, and call FRT_GRAPH_INIT before any other FRTLIB routines.

NOTE

> The original library LSL$LIBRARY:FRTLIB that referenced VIOLIB and CMDLIB has been superceded by the current library LSLFRTLIB that makes calls to LSLLIB. This library should no longer be used - the common blocks supplied with LSLFRTLIB do not match those used by it.
>
> LSLLIB must be initialised with a call of the subroutine LSL_INIT (see section 5, below).

If the plotting routines are used, then graphics routines must be supplied. By default, the library references GKS (Graphical Kernel System) routines.

3 *GRAPHICS INTERFACE*

FRTLIB includes routines to plot symbols, texts and patterned lines. By default, the library references the following GKS routines - GQLWSC, GQPLCI, GQLN, GSLWSC, GSPLCI, GSLN and GPL for polylines, GQFAIS, GQFASI, GQFACI, GSFAIS, GSFASI and GFA for fill areas, GQMCI, GQMK, GSMCI, GSMK and GPM for symbols drawn by hardware, GSTXFP, GSTXCI, GSCHH, GSCHUP and GTX for texts drawn by hardware. GESC, GGDP (for hardware circles and curves), and (LSL routines) LSG_SET_PATTERN, LSG_BEGIN_AREA, LSG_END_AREA, LSG_STRING_WIDTH, and LSG_STRING_BOUNDS are also used in a way which is unlikely to be consistent with arbitrary GKS systems.

If a true GKS system is used, then it is the responsibility of the calling program to set it up correctly before calling FRTLIB. In particular, GKS must be open, with the desired workstation(s) open and active. All aspect source flags should be set to INDIVIDUAL, and any desired transformations should be set up.

FRTLIB restores the various GKS attributes to their previous values after plotting, except the various text attributes (it is expected that FRTLIB will be entirely responsible for plotting text). If the workstation supports thick lines, then its nominal line thickness must be communicated to FRTLIB by a call to SRISLW (q.v.). Thick lines may be suppressed completely by setting the logical variable FRTHKS in common FRTCOM to .TRUE. If the symbols and texts being plotted include circle arcs, interpolated curves or fill areas or if PATGPL is used to draw patterned lines then a call to SRIUNI (q.v.) should be made to give FRTLIB information about the units being used and any rotation that has been applied to coordinates relative to the GKS window by the calling program. The default values are appropriate for a drawing scale such that one unit corresponds to one millimetre.

In the present implementation, the graphical characteristics used when plotting texts or symbols, are taken from the SCT entries for the individual components of the symbol/character. If these fields are absent or invalid (usually = 0) then the values in the FRT entry for the complete feature are used. If these are also missing or invalid, then defaults are used.
The graphical characteristics that are being referred to are the line width, cross hatching spacing (for fill areas), hardware line type, drawing tool and colour.
Routine SRICOL may be used to specify the symbols and texts are to be drawn in a particular overriding colour.

The GKS routines may be supplied by the user if desired, rather than using a full GKS system. The calls to GKS are contained within nineteen routines in FRTLIB, the specification of which follows, and the user may alternatively replace these.

It is possible to use FRTLIB to plot on three dimensional devices. This is achieved by calls to SRI_SET_Z and use of the Z arguments to SRIGPL, SRIGFA, SRIGPM, SRIGTX, SRIGDP and PATGPL (see below).

3.1 *SRIGQP - Inquire Polyline Attributes*

```
        SUBROUTINE SRIGQP(WIDTH,ICOL)
        Returns REAL    WIDTH - linewidth scale factor
                INTEGER ICOL  - polyline colour index
```

This routine is used to preserve the attributes, so that they may be reset by a later call to SRIGSP.

3.2 *SRIGSP - Set Polyline Attributes*

```
        SUBROUTINE SRIGSP(WIDTH,ICOL)
                REAL    WIDTH - linewidth scale factor
                INTEGER ICOL  - polyline colour index
```

This routine is used to set the polyline attributes. It may be passed WIDTH=0.0, meaning normal thickness. Zero is an invalid linewidth scale factor in GKS, and SRIGSP must detect this case.

3.3 *SRIGPL - Polyline*

```
        SUBROUTINE SRIGPL(N,X,Y,Z)
                INTEGER N     - number of points
                REAL    X(N)  - x coordinates
                REAL    Y(N)  - y coordinates
                REAL    Z(N)  - z coordinates
```

This routine is used to draw lines. It should draw a line connecting the given points.

As supplied in FRTLIB, this makes a call to the GKS routine GPL and the 4th argument is not required. However, this routine is designed to be replaced, when required, by any application program, and when this is done the 4th argument is used to pass the heights of the points.

3.4 *SRIGQA - Inquire Fill Area Attributes*

```
        SUBROUTINE SRIGQA(STYLE,INDEX,COLOUR,WIDSEP)
        Returns INTEGER STYLE     - fill area internal style
                INTEGER INDEX     - fill area style index
                INTEGER ICOL      - fill area colour index
                REAL    WIDSEP(2) - line width and separation
                                  - for hatching
```

This routine is used to preserve the attributes, so that they may be reset by a later call to SRIGSA.

3.5 *SRIGSA - Set Fill Area Attributes*

```
        SUBROUTINE SRIGSA(STYLE,INDEX,COLOUR,WIDSEP)
                INTEGER STYLE     - fill area interior style
                INTEGER INDEX     - fill area style index
                INTEGER ICOL      - fill area colour index
                REAL    WIDSEP(2) - line width and separation
                                  - for hatching
```

This routine is used to set the fill area attributes.

3.6 *SRIGFA - Fill Area*

```
        SUBROUTINE SRIGFA(N,X,Y,Z)
                INTEGER N     - number of points
                REAL    X(N)  - x coordinates
                REAL    Y(N)  - y coordinates
                REAL    Z(N)  - z coordinates
```

This routine is used to draw fill areas. It should draw the appropriate area using the fill style and index specified by SRIGSA. If the polygon is not closed, then this routine should close it.

As supplied in FRTLIB, this makes a call to the GKS routine GFA and the 4th argument is not required. However, this routine is designed to be replaced, when required, by any application program, and when this is done the 4th argument is used to pass the heights of the points.

3.7 *SRIGQM - Inquire Polymarker Attributes*

```
        SUBROUTINE SRIGQM(SYMNO,ANGLE,ICOL)
        Returns INTEGER SYMNO   - symbol
                REAL    ANGLE   - angle to draw it at
                INTEGER ICOL    - colour to use
```

This routine is used to preserve the attributes, so that they may be reset by a later call to SRIGSM.

3.8 *SRIGSM - Set Polymarker Attributes*

```
        SUBROUTINE SRIGSM(SYMNO,ANGLE,ICOL)
                INTEGER SYMNO   - symbol
                REAL    ANGLE   - angle to draw it at
                INTEGER ICOL    - colour to use
```

This routine is used to set the polymarker attributes.

3.9 *SRIGPM - Polymarker*

```
        SUBROUTINE SRIGPM(N,X,Y,Z)
                INTEGER N     - number of points
                REAL    X(N)  - x coordinates
                REAL    Y(N)  - y coordinates
                REAL    Z(N)  - z coordinates
```

This routine is used to draw polymarkers. It should draw the current polymarker at each of the given points.

As supplied in FRTLIB, this makes a call to the GKS routine GPM and the 4th argument is not required. However, this routine is designed to be replaced, when required, by any application program, and when this is done the 4th argument is used to pass the heights of the points.

3.10 *SRIGST - Set Text Attributes*

```
        SUBROUTINE SRIGST(FONT,COLOUR,SIZE,ANGLE)
                INTEGER FONT    - font number
                INTEGER COLOUR  - colour to use
                REAL    SIZE    - text height in world units
                REAL    ANGLE   - angle to draw it at
```

This routine is used to set the hardware text attributes.

3.11 *SRI_STRING_WIDTH - Inquire Hardware Text Width*

```
        LOGICAL FUNCTION SRI_STRING_WIDTH(STRING,FONT,WIDTH)
                CHARACTER*(*)   STRING  - character string
                INTEGER         FONT    - font number
                REAL            WIDTH   - returned width
```

This routine is used to get the width of a hardware text string (if drawn at unit height in the given font). By default it calls LSG_STRING_WIDTH. It should return .TRUE. if it cannot obtain the string width.

3.12 *SRI_STRING_BOUNDS - Inquire Hardware Text Boundary*

```
        LOGICAL FUNCTION SRISTRINGBOUNDS(STRING,FONT,BORDER,ROUTINE)
                CHARACTER*(*)   STRING  - character string
                INTEGER         FONT    - font number
                REAL            BORDER  - border round characters
                EXTERNAL        ROUTINE - callback routine
```

This routine is used to get a bounding region for a hardware text string (if drawn at unit height in the given font). By default it calls LSG_STRING_BOUNDS. It should return .TRUE. if it is unable to return the information. The information is returned be calling

```
        ROUTINE(WIDTH,LLX,LLY,URX,URY)
```

for each character in the string.

3.13 *SRIGTX - Text*

```
        SUBROUTINE SRIGTX(X,Y,Z,STRING)
                REAL            X  - x coordinates
                REAL            Y  - y coordinates
                REAL            Z  - z coordinates
                CHARACTER*(*)   STRING - character string
```

This routine is used to draw hardware text. It should draw the text string at the given point.

As supplied in FRTLIB, this makes a call to the GKS routine GTX and the 3rd argument is not required. However, this routine is designed to be replaced, when required, by any application program, and when this is done the 3rd argument is used to pass the height of the points.

3.14 *SRIGQL - Inquire Drawing Hardware*

```
        SUBROUTINE SRIGQL(LINETYPE,HARDWARE,ANGLE)
        Returns INTEGER LINETYPE        - line type
                INTEGER HARDWARE        - hardware tool to be used for line
                REAL    ANGLE           - angle tool is set at
```

This routine is used to preserve the attributes, so that they may be reset by a later call to SRIGSL.

3.15 *SRIGSL - Set Drawing Hardware*

```
        SUBROUTINE SRIGSL(LINETYPE,HARDWARE,ANGLE)
                INTEGER LINETYPE        - line type
                INTEGER HARDWARE        - hardware to be used for line
                REAL    ANGLE           - angle tool is set at
```

This routine is used to set the hardware for drawing.

3.16 *SRIGDP - Generalised Drawing Primitive*

```
        SUBROUTINE SRIGDP(N,X,Y,Z,ID)
                INTEGER N               - number of coords
                REAL    X(N)    - x coordinates
                REAL    Y(N)    - y coordinates
                REAL    Z(N)    - z coordinates
                INTEGER ID      - GDP id.
```

This arguments are passed on to GKS routine GGDP. ID controls what coordinates are expected.

```
        ID = 1 full circle, 2 points, centre, edge
        ID = 2 clockwise arc, 3 points, start, end, centre
        ID = 3 anti-clockwise arc, 3 points, start, end, centre
        ID = 6 interpolated curve connecting the points
```

As supplied in FRTLIB, this makes a call to the GKS routine GGDP and the 4th argument is not required. However, this routine is designed to be replaced, when required, by any application program, and when this is done the 4th argument is used to pass the heights of the points.

3.17 *SRI_SET_PATTERN - Set Up Hardware Pattern*

```
        INTEGER FUNCTION SRISETPATTERN(LEN,MAJ,MIN,MAREP,MIREP,FLG)
                REAL    LEN     - overall length
                REAL    MAJ     - major dash length
                REAL    MIN     - minor dash length
                INTEGER MAREP   - major repeat count
                INTEGER MIREP   - minor repeat count
                INTEGER FLG     - pattern flags
```

The arguments are passed on to LSG_SET_PATTERN. If the specified pattern can be drawn by hardware, the function returns a number which will subsequently be passed as the LINE_TYPE argument to GSLN via SRIGSL. Otherwise return 0.

3.18 *SRI_BEGIN_AREA - Begin A Composite Area*

```
        SUBROUTINE SRI_BEGIN_AREA
```

This calls LSG_BEGIN_AREA. Subsequent calls to SRIGFA are to be treated as part of a composite area, until a call to SRI_END_AREA.

3.19 *SRI_END_AREA - End A Composite Area*

```
        SUBROUTINE SRI_END_AREA
```

This calls LSG_END_AREA. It indicates the end of a composite area, which should now be filled as appropriate.

4 *COMMON BLOCKS*

The common blocks of the FRT system are held in LSL$CMNFRT:, and can be included into user programs by using INCLUDE statements. The ones which may be used by user programs are:-

FRTCOM.CMN is the main common block holding the FRT table itself, together with information about the currently selected Feature Code.

FRTPRIO.CMN holds the priority definition table.

FRTFIL.CMN holds the area fill pattern definition table.

FRTGRP.CMN holds the Feature Code Group definitions.

FRTPAT.CMN holds the line pattern definition table.

FRTSCT.CMN is similar to FRTCOM.CMN, but holds information about the Symbol Component Table (SCT).

FRTSRI.CMN holds the Symbol Representation Index table (SRI).

FRTTRI.CMN is similar to FRTSRI.CMN, but holds the Text Representation Index table (TRI).

TRIEXT.CMN holds character extent with respect to the locating point

FRTACD.CMN holds the Attribute Code Definitions.

These common blocks contain several variables whose names end with "_LOC". These contain the addresses of memory obtained at run time containing arrays of data read from the FRT files. It should not normally be necessary for programs to reference these arrays, but if they must be accessed, then one method (in Fortran, and taking FRTINT_LOC as an example) is to pass %VAL(FRTINT_LOC) as an argument to a subroutine, and within the subroutine declare the argument as INTEGER*2 FRTINT(6,FRTMAX). The array may then be accessed just as the old FRTINT array was in previous releases of the library. References to the group command table should now use %VAL(GRPCMT_LOC), rather then just GRPCMT as previously. Access to the group bitmaps should now be made using the routines GRPFCT and GRPFC.

4.1 *FRTCOM*

```
C
C FRT library interface main common block FRTCOM.CMN
C holds Feature Representation Table itself,
C the current selected FC and various useful parameters
C
C
Cmod    add FRTFLG, FRTHW and two more cols to FRTINT
C       make FRTINT I*2                         RWR     19-May-1985
Cmod    add FRTAST, FRTAIX                      TJI      3-Dec-1984
Cmod    add FRTARE, move logicals to end        TJI     14-Nov-1984
Cmod    PARAMETER ARETYP added                  RWR     13-Nov-l984
C
C
        PARAMETER FRTMAX_DEF=1000       ! def number of FCs
C
C Define all the graphical types as parameters
C
        PARAMETER LINTYP = 1            ! line string
        PARAMETER CLOTYP = 2            ! clockwise circle arc
        PARAMETER ANTTYP = 3            ! anti-clockwise circle arc
        PARAMETER CIRTYP = 4            ! circum-circle arc
        PARAMETER FULTYP = 5            ! full circumcircle
        PARAMETER CURTYP = 6            ! interpolated curve
        PARAMETER UNOTYP = 7            ! unoriented symbol
        PARAMETER ORITYP = 8            ! oriented symbol
        PARAMETER SCATYP = 9            ! scaled symbol
        PARAMETER TEXTYP = 10           ! text
        PARAMETER STRTYP = 11           ! symbol string
        PARAMETER ARETYP = 12           ! fill area
C
        INTEGER         FRTMAX          ! number of FCs
C
C FC selection control and attributes of selected FC
C
        INTEGER*4       FRTCNT          ! count of defined FCs
        INTEGER*4       FRTIND          ! index of selected FC
        INTEGER*4       FRTFC           ! the selected FC
        INTEGER*4       FRTGT           ! its Graphical Type
        INTEGER*4       FRTCOL          ! its colour
        REAL            FRTWID          ! its width
        REAL            FRTSIZ          ! its size
        INTEGER*4       FRTSC           ! its Secondary Code
        INTEGER*4       FRTFLG          ! flags word
C
        INTEGER*4       FRTHW           ! symbol for hardware line
        INTEGER*4       FRTHWL          ! hardware line style
C
        INTEGER*4       FRTAST          ! fill area internal style
        INTEGER*4       FRTAIX          ! fill area style index
C
        LOGICAL*4       FRTHWS          ! true if to use hardware symbol
C
        LOGICAL*4       FRTLIN          ! true if linear
        LOGICAL*4       FRTSYM          ! true if symbol
        LOGICAL*4       FRTARC          ! true if circle arc
        LOGICAL*4       FRTCUR          ! true if curve
        LOGICAL*4       FRTTEX          ! true if text
        LOGICAL*4       FRTARE          ! true if fill area
C
C the main FRT table
C
C pointer to array of INTEGER*2 (6,FRTMAX)
        INTEGER*4       FRTINT_LOC      ! ptr to integers
C
C pointer to array of REAL*4 (2,FRTMAX)
        INTEGER*4       FRTFLT_LOC      ! ptr to floats (reals)
C
C global control variables
C
        LOGICAL*4       FRTHKS          ! true if thick lines supressed
        LOGICAL*4       FRTCLP          ! true if to clip symbols in
                                        ! patterened fill areas
C
C
        COMMON/FRTCOM/FRTMAX,FRTCNT,FRTIND,FRTFC,FRTGT,FRTCOL,
     &                FRTWID,FRTSIZ,FRTSC,FRTFLG,FRTHW,FRTHWL,
     &                FRTAST,FRTAIX,FRTHWS,
     &                FRTLIN,FRTARC,FRTCUR,FRTSYM,FRTTEX,FRTARE,
     &                FRTHKS,FRTCLP,
     &                FRTINT_LOC,FRTFLT_LOC
```

```
C
```

4.2 *FRTFIL*

```
C
C FRT library interface subsidiary common block FRTFIL.CMN
C defines patterns for area fill with patterned lines
C
C define limits etc
C
        PARAMETER       FILMAX_DEF = 100! def no of fill patterns
C
        INTEGER*4       FILMAX          ! max no of patterns
        INTEGER*4       FILCNT          ! how many defined
        INTEGER*4       FILIND          ! current pattern fill index
C
C the selected pattern and its atributes
C
        INTEGER*4       FILSEL          ! selected pattern fill no
        INTEGER*4       FILPAT          ! hatch direction
        INTEGER*4       FILSC           ! line pattern no
C
C now the main arrays
C
C pointer to array of INTEGER*2 (3,FILMAX)
        INTEGER*4       FILINT_LOC      ! ptr to integer parts
C
        COMMON/FRTFIL/FILMAX,FILCNT,FILIND,FILSEL,FILPAT,FILSC,
     &                FILINT_LOC
C
```

4.3 *FRTGRP*

```
C
C FRT library interface subsidiary common block FRTGRP.CMN
C Defines data structures to hold GROups of FCs
C
        PARAMETER       GRPMAX_DEF = 30         ! def no of groups
        PARAMETER       GRPMXC = 32768          ! max FCs (0-32767)
C
        INTEGER*4       GRPCNT                  ! no of defined groups
        INTEGER*4       GRPMAX                  ! max no of groups
C
C Pointer to an LSLLIB command table with room for GRPMAX commands
        INTEGER*4       GRPCMT_LOC              ! ptr to command table
C
C Pointer to array of bits (GRPMXC,GRPMAX)
        INTEGER*4       GRPFCT_LOC              ! ptr to bitmap of FCs
C
        INTEGER*4       GRPSAV(12)              ! to save command table
C
```

```
        COMMON/FRTGRP/GRPMAX,GRPCNT,GRPCMT_LOC,GRPFCT_LOC,GRPSAV
C
```

4.4 *FRTPAT*

```
C
C FRT library interface subsidiary common block FRTPAT.CMN
C defines patterns for fancy line generation
C
C define limits etc
C
        PARAMETER       PATLIM_DEF = 100        ! def no of patterns
C
        INTEGER*4       PATLIM                  ! max no of patterns
        INTEGER*4       PATCNT                  ! how many defined
        INTEGER*4       PATIND                  ! current pattern index
C
C the selected pattern and its atributes
C
        INTEGER*4       PATSEL                  ! selected pattern no
        INTEGER*4       PATMAJ                  ! major subunit
        INTEGER*4       PATMIN                  ! minor subunit
        INTEGER*4       PMAREP                  ! major repeat count
        INTEGER*4       PMIREP                  ! minor repeat count
        INTEGER*4       PATFLG                  ! flags word
        REAL            PATSIZ                  ! overall size
        REAL            PMASIZ                  ! major size
        REAL            PMISIZ                  ! major size
        REAL            PMAWID                  ! major width
        REAL            PMIWID                  ! major width
        REAL            PATOFF                  ! offset
C
C now the main arrays
C
C pointer to array of INTEGER*2 (6,PATLIM)
        INTEGER*4       PATINT_LOC              ! ptr to integer parts
C
C pointer to array of REAL*4 (6,PATLIM)
        INTEGER*4       PATDIM_LOC              ! ptr to real parts
C
        COMMON/FRTPAT/PATLIM,PATCNT,PATIND,PATSEL,
     &                PATMAJ,PATMIN,PMAREP,PMIREP,PATFLG,
     &                PATSIZ,PMASIZ,PMISIZ,PMAWID,PMIWID,PATOFF,
     &                PATINT_LOC,PATDIM_LOC
C
```

4.5 *FRTPRIO*

```
C
C FRT library interface subsidiary common block FRTPRIO.CMN
C defines represenions and priorities for multipass drawing
```

```
C
C define limits etc
C
        PARAMETER       PRIOLIM_DEF = 20        ! def no of priority records
        PARAMETER       PRIO_PER_FC_MAX = 8     ! max number of fc-rep pairs
        PARAMETER       PRIO_VALUE_MAX = 32767  ! largest priority allowed
        PARAMETER       PRIO_FC_MAX = 32767     ! largest feature code allowed
        PARAMETER       PRIO_DEFAULT_DEFAULT = 3 ! default default value
C
C the selected priority record (returned by call of FRTPRIOFND)
C
        INTEGER*4       PRIO_SEL                ! selected fc
        INTEGER*4       PRIO_REP(PRIO_PER_FC_MAX) ! list of representations
        INTEGER*4       PRIO_PRIO(PRIO_PER_FC_MAX)! list of priorities
        INTEGER*4       PRIO_NUMBER             ! number of representations
                                                ! in PRIO_REP and PRIO_PRIO
C
C other values
        INTEGER*4       PRIO_MAX                ! maximum priority encountered
        INTEGER*4       PRIO_DEFAULT            ! default priority for feature
                                                ! codes not in priority table
C
C ***************************************************************************
C
C data used internally by FRTLIB
C
        INTEGER*4       PRIOLIM                 ! max no of priority records
        INTEGER*4       PRIOCNT                 ! how many defined
        INTEGER*4       PRIOIND                 ! current priority record
C
C
C a bitmap of the priorities that have been used
        PARAMETER       PRIO_PRBM_SIZE = PRIO_VALUE_MAX/32+1
        INTEGER*4       PRIOPRBMAP(PRIO_PRBM_SIZE)
C
C now the main arrays
C
C pointer to array of INTEGER*2 (2*FRT_PRIO_PER_FC_MAX+1,PRIOLIM)
        INTEGER*4       PRIOINT_LOC             ! ptr to integer parts
C
        COMMON/FRTPRIO/PRIOPRBMAP,PRIOINT_LOC,PRIOLIM,PRIOCNT,PRIOIND,
     &                 PRIO_MAX,PRIO_DEFAULT,PRIO_SEL,PRIO_NUMBER,
     &                 PRIO_REP,PRIO_PRIO
C
```

4.6 *FRTSCT*

```
C
C FRT library interface subsidiary common block FRTSCT.CMN
C holds the Symbol Component Table (SCT)
C this common block follows the same structure as the FRT in FRTCOM.CMN
C
        PARAMETER SCTMAX_DEF=200        ! def number of SCTs
C
```

```
        INTEGER*4       SCTCNT          ! count of defined SCs
        INTEGER*4       SCTIND          ! index of selected SC
        INTEGER*4       SCTCC           ! the selected code
        INTEGER*4       SCTGT           ! its Graphical Type
        INTEGER*4       SCTCOL          ! its colour
        REAL            SCTWID          ! its width
        REAL            SCTSIZ          ! its size
        INTEGER*4       SCTSC           ! its Secondary Code
        INTEGER*4       SCTFLG          ! flags word
        INTEGER*4       SCTHW           ! hardware line
        INTEGER*4       SCTHWL          ! hardware line style
        LOGICAL*4       SCTHWS          ! true if to use hardware symbol
C
        INTEGER         SCTMAX          ! number of SCTs
C
C from the value in SCTSC, we can deduce (for fill areas)
C
        INTEGER*4       SCTAST          ! internal style
        INTEGER*4       SCTAIX          ! style index
C
C and the arrays which hold the actual data about all of the components
C
C pointer to array of INTEGER*2 (6,SCTMAX)
        INTEGER*4       SCTINT_LOC      ! integers
C
C pointer to array of REAL*4 (2,SCTMAX)
        INTEGER*4       SCTFLT_LOC      ! floats (reals)
C
        COMMON/FRTSCT/SCTMAX,SCTCNT,SCTIND,SCTCC,SCTGT,SCTCOL,
     &               SCTWID,SCTSIZ,SCTSC,SCTFLG,SCTHW,
     &               SCTHWL,SCTHWS,SCTAST,SCTAIX,
     &               SCTINT_LOC,SCTFLT_LOC
C
```

4.7 *FRTSRI*

```
C
C FRT library interface SRI common block FRTSRI.CMN
C holds Symbol Representations from SRI file,
C the current selected symbol and various useful parameters
C
        PARAMETER SRIMAX_DEF=7000       ! def size of SRI table
C
C symbol selection control and attributes of selected symbol
C
        INTEGER*4       SRIMAX          ! maximum size of SRI table
        INTEGER*4       SRICNT          ! count of defined symbols
        INTEGER*4       SRIIND          ! index of selected symbol
        INTEGER*4       SRITOP          ! top of used buffer
        INTEGER*4       SRISEL          ! the selected symbol
        LOGICAL         SRIHWS          ! .true. if hardware symbols
                                        ! available
        LOGICAL         SRIHWC          ! .true. if hardware circles
```

```
                                        ! available
        LOGICAL         SRIHWP          ! .true. if hardware patterns
                                        ! available
        INTEGER         SRIHWL          ! no. of hardware line types
                                        ! available
        LOGICAL         SRIHWI          ! .true. if hardware curves
                                        ! (interpolation) available
C
C the main SRI table
C
C Pointer to array of REAL*4 (2,SRIMAX)
        INTEGER*4       SRIBUF_LOC      ! ptr to coord array
C
C Pointer to array of INTEGER*2 (SRIMAX)
        INTEGER*4       SRITAB_LOC      ! ptr to symbol and SCT numbers
C
C
        COMMON/FRTSRI/SRIMAX,SRICNT,SRIIND,SRITOP,SRISEL,
     &                SRIHWS,SRIHWC,SRIHWP,SRIHWL,SRIHWI,
     &                SRIBUF_LOC,SRITAB_LOC
C
```

4.8 *FRTTRI*

```
C
C FRT library interface TRI common block FRTTRI.CMN
C holds Text Representation Index from TRI file, a table of widths,
C the current selected character and various useful parameters
C
        PARAMETER TRIMAX_DEF=10000      ! def size of TRI table
        PARAMETER TRIMXC=255            ! number of chars in a font
        PARAMETER TRIMXF_DEF=5          ! def number of fonts
C
C constants for italic text transformation
C
        REAL    ITALIC_A1,ITALIC_A2,ITALIC_B1,ITALIC_B2
C
        PARAMETER       (ITALIC_A1 = 1.0)
        PARAMETER       (ITALIC_A2 = 0.5)
        PARAMETER       (ITALIC_B1 = 0.0)
        PARAMETER       (ITALIC_B2 = 1.0)
C
C maxima
        INTEGER         TRIMAX          ! maximum size of TRI table
        INTEGER         TRIMXF          ! maximum number of fonts
C
C symbol selection control and pointers
C
        INTEGER*4       TRICNT          ! count of defined characters
        INTEGER*4       TRIIND          ! index of selected characters
        INTEGER*4       TRITOP          ! top of used buffer
        INTEGER*4       TRISEL          ! the selected characters
C
C the main TRI table
```

```
C
C Pointer to array of REAL*4 (2,TRIMAX)
        INTEGER         TRIBUF_LOC      ! ptr to coords
C
C Pointer to array of INTEGER*2 (TRIMAX)
        INTEGER         TRITAB_LOC      ! ptr to characters and SCT numbers
C
C the width table
C Pointer to array of REAL*4 (TRIMXC,TRIMXF)
        INTEGER         TRIWID_LOC      ! ptr to widths for spacing
C
C font control
C
        INTEGER*4       TRIFNC          ! count of defined fonts
C
C Pointer to array of INTEGER*2 (TRIMXF)
        INTEGER         TRIFNT_LOC      ! ptr to font numbers
C
C Pointer to array of INTEGER*4 (TRIMXF)
        INTEGER         TRIFNP_LOC      ! ptr to font pointers
C
C position of plotted letter (required for transformation to italic)
C
        REAL*4          TRIPOSX,TRIPOSY
        REAL*4          TRIANG
C
C and transformation to use for italic letters
C
        REAL*4          TRIA1,TRIA2,TRIB1,TRIB2
C
C and whether composite characters (e.g. {Zcaron}) in use
C
        LOGICAL         TRICC
C
        COMMON/FRTTRI/TRIMAX,TRICNT,TRIIND,TRITOP,TRISEL,
     *                TRIFNC,TRIMXF,TRIPOSX,TRIPOSY,
     *                TRIA1,TRIA2,TRIB1,TRIB2,TRIANG,
     *                TRIBUF_LOC,TRITAB_LOC,TRIWID_LOC,
     *                TRIFNT_LOC,TRIFNP_LOC,TRICC
C
```

4.9 *TRIEXT*

```
C
C FRT library interface subsidiary common block TRIEXT.CMN
C holds character extent with respect to the locating point
C
C flags for signalling to SRI_LINE and TRI_EXTENT
C
        LOGICAL         GET_EXTENT      ! get extent, don't plot
        LOGICAL         START_IT        ! initialise for new character
C
C maxima and minima
C
```

```
        REAL            MIN_X_EXT       ! minimum X
        REAL            MAX_X_EXT       ! maximum X
        REAL            MIN_Y_EXT       ! minimum Y
        REAL            MAX_Y_EXT       ! maximum Y
C
        COMMON/TRIEXT/GET_EXTENT,START_IT,
     &  MIN_X_EXT,MAX_X_EXT,MIN_Y_EXT,MAX_Y_EXT
C
```

4.10 *FRTACD*

```
C
C FRT library interface subsidiary common block FRTACD.CMN
C
C The number of user ACDs (in addition to the LSL default ones) is
C taken from logical name LSL$FRT_ACDMAX (range 0-32767). If this
C is not set up, or is invalid, then a default of ACD_USER is used.
        PARAMETER       ACD_USER = 50           ! default user ACDs
C
        PARAMETER       ACDOFFSET = 1000        ! offset of each table
        PARAMETER       ACD_CODE_MAX = 32767    ! maximum allowed CODE
C
        PARAMETER       ACD_FORMAT_MAX = 8      ! max format length
        PARAMETER       ACD_NAME_MAX = 20       ! max name length
C
C ACD data types
        INTEGER         ACD_DATATYPE_I
        INTEGER         ACD_DATATYPE_R
        INTEGER         ACD_DATATYPE_C
        INTEGER         ACD_DATATYPE_D
        INTEGER         ACD_DATATYPE_T
C
        PARAMETER       (ACD_DATATYPE_I = 1)    ! integer
        PARAMETER       (ACD_DATATYPE_R = 2)    ! real
        PARAMETER       (ACD_DATATYPE_C = 3)    ! 4 characters
        PARAMETER       (ACD_DATATYPE_D = 4)    ! date
        PARAMETER       (ACD_DATATYPE_T = 5)    ! time
C
        INTEGER*4       ACD_DEF_MINI            ! default values for
        INTEGER*4       ACD_DEF_MAXI            ! min and max values
        REAL*4          ACD_DEF_MINR
        REAL*4          ACD_DEF_MAXR
        INTEGER         ACD_DEF_MINC
        INTEGER         ACD_DEF_MAXC
        CHARACTER*(*)   ACD_DEF_MIND
        CHARACTER*(*)   ACD_DEF_MAXD
        CHARACTER*(*)   ACD_DEF_MINT
        CHARACTER*(*)   ACD_DEF_MAXT
        PARAMETER       (ACD_DEF_MINI = -2147483647)
        PARAMETER       (ACD_DEF_MAXI =  2147483647)
        PARAMETER       (ACD_DEF_MINR = -1.0E37)
        PARAMETER       (ACD_DEF_MAXR =  1.0E37)
        PARAMETER       (ACD_DEF_MINC = '    ')
        PARAMETER       (ACD_DEF_MAXC = '~~~~')
```

```
        PARAMETER       (ACD_DEF_MIND = '17-NOV-1858')
        PARAMETER       (ACD_DEF_MAXD = '31-DEC-9999')
        PARAMETER       (ACD_DEF_MINT = '00:00:00.00')
        PARAMETER       (ACD_DEF_MAXT = '23:59:59.99')
C
C Attributes of selected AC
C
        INTEGER*4       ACD_CODE                 ! code
        INTEGER*4       ACD_DATA_TYPE            ! data type
        CHARACTER*(ACD_FORMAT_MAX) ACD_FORMAT    ! format
        INTEGER*4       ACD_FORMAT_LEN           ! length of format
        INTEGER*4       ACD_MIN_MAX_I(2)         ! min max (integer)
        REAL*4          ACD_MIN_MAX_R(2)         ! min max (real)
        CHARACTER*(ACD_NAME_MAX) ACD_NAME        ! full name; space filled
        INTEGER*4       ACD_NAME_LEN             ! its length
        REAL*4          ACD_INTERVAL             ! its granularity
C
        EQUIVALENCE     (ACD_MIN_MAX_I,ACD_MIN_MAX_R)
C
        INTEGER*4       ACDCMT                   ! %LOC(command table)
C
        COMMON/FRTACD/ACD_CODE,ACD_DATA_TYPE,ACD_FORMAT_LEN,
     &                 ACD_MIN_MAX_I,ACD_NAME_LEN,ACD_INTERVAL,ACDCMT

        COMMON/FRTACDC/ACD_NAME,ACD_FORMAT
C
```

5 *SUBROUTINES*

Most of the routines in the FRT library are defined as FORTRAN logical functions returning .FALSE. if they succeed, and .TRUE. if they fail.

Before any call to FRTLIB subroutines are made, the library LSLLIB must be initialised by a call to the subroutine LSL_INIT. Programs that use FRTLIB, but do not otherwise access LSLLIB must ensure that a call is made to this routine before any calls to FRTLIB routines are made. LSL_INIT is in LSLLIB, but its specification is included below, as it is necessary for the use of (LSL)FRTLIB.

Programs which use the shareable image version of FRTLIB to perform graphics may call the routine FRT_GRAPH_INIT as a convenient way of setting up the addresses of the graphics routine in common block SRIVEC.

All programs using the FRT library must either call routine FRTINI, passing it the name of an FRT file as argument, or if only the attribute code definition routines and the default attribute code definitions are required, the routine FRT_ACDINI. If the former case then the FRT file has been read into the various common blocks, and routines such as FRTFND can be called to find the representation of a given Feature Code. In the latter case, the ACD common blocks are filled with the default attribute code information. The information about attribute definitions are accessed by the routines ACDFND and ACDFND_NAME.

If symbol and character handling is needed then routines SRIINI and TRIINI must be called before any other SRI or TRI routines may be used.

5.1 LSL_INIT

```
        call LSL_INIT( [timer] )
```

This routine initialises the library LSLLIB. If the argument timer is true or absent, then an exit handler is set up which will cause timing statistics to be output by the program when it exits.

5.2 FRT_GRAPH_INIT

```
        call FRT_GRAPH_INIT
```

This routine (in LSLFRTLIB.OLB, but not FRTSHR.EXE) may be called as a convenient method of setting the addresses of the nineteen graphics routines in common SRIVEC. These must be set before attempting to perform any graphics. FRT_GRAPH_INIT loads the addresses of a set of routines with the default names (e.g. SRIGPL) which are also contained in LSLFRTLIB.OLB.

5.3 *FRTINI*

```
        failed = FRTINI(frtfile)
```

eg

```
        IF (FRTINI('LSL$FRT:FRT.FRT')) THEN failed to read file
```

This routine MUST be called before any other FRT library routines. It opens the given file and obeys the commands within it to set up the FRT, SCT, priority, pattern, area fill, group and ACD common blocks.

NOTE

The ACD common block is filled with the LSL default values, even if there are no ACD entries in the FRT file.

This is similar to having the following ACD commands in the file

```
        ACD I  1 Secondary_FC     0           32767
        ACD I  2 Contour         -2147483647 2147483647

        ACD R  3 Height          -1.0E37     1.0E37
        ACD I  4 LH_boundary       0           32767
        ACD I  5 RH_boundary       0           32767
        ACD I  6 Text              0           32767
        ACD I  7 DFAD_FADT         0           0
        ACD I  8 DFAD_ACC          0           0
        ACD I  9 Parent_FSN        0           65535
        ACD I 10 RELHT_START       0           100
        ACD I 11 RELHT_END         0           100
        ACD R 80 Cliff_left        -1.0E37     +1.0E37
        ACD R 81 Cliff_right       -1.0E37     +1.0E37
        ACD R 82 Polygon_info      -1.0E37     +1.0E37
        ACD R 91 X                 -1.0E37     +1.0E37
        ACD R 92 Y                 -1.0E37     +1.0E37
        ACD R 93 Z                 -1.0E37     +1.0E37
        ACD R 94 ZB                -1.0E37     +1.0E37
        ACD R 95 ZC                -1.0E37     +1.0E37
        ACD R 96 ZD                -1.0E37     +1.0E37
        ACD R 97 Dheight           -1.0E37     +1.0E37
```

After a FRT file has been read with this command the variables PRIO_MAX and PRIO_DEFAULT are set in the common block LSL$CMNFRT:FRTPRIO.CMN. PRIO_MAX is the highest priority that has been set with PRIORITY records in the FRT file. It does not include PRIO_DEFAULT, which is the priority to be associated with all feature codes that do not occur in PRIORITY records.

5.4 *FRTFND*

```
        failed = FRTFND(fc,[output_error])
```

eg

```
        IF (FRTFND(23)) THEN failed to find Feature Code 23
```

This routine is called to find the representation of a given feature code. It sets up variables in COMMON/FRTCOM/. If the optional argument "output_error" is .FALSE. then the error message which would normally be output when "fc" is not found in the FRT table is suppressed.

5.5 *PATFND*

```
        failed = PATFND(patno)
eg
        IF (PATFND(12)) THEN failed to find pattern 12
```

This routine is called to find the definition of a given pattern. It sets up variables in COMMON/FRTPAT/.

5.6 *FILFND*

```
        failed = FILFND(filno)
eg
        IF (FILFND(-4)) THEN failed to find area fill pattern -4
```

This routine is called to find the definition of a given area fill pattern. It sets up variables in COMMON/FRTFIL/.

5.7 *FRTFGT*

```
        failed = FRTFGT(gt,fc)
eg
        IF (FRTFGT(10,FC)) THEN failed to find graphical type 10
```

This routine is called to return in FC the first feature code in the FRT with graphical type GT.

5.8 FRT_ACDINI

```
        failed = FRT_ACDINI()

eg
        IF (FRT_ACDINI()) THEN failed
```

This routine must be called before any other ACD library routines, if FRTINI has not been called. It makes the default set of attribute code definitions (see above) available to ACDFND and ACDFND_NAME.

Note that this routine is called by FRTINI, so if it is called after FRTINI then any ACD definitions contained in the FRT file will be lost.

5.9 *ACDFND*

```
        failed = ACDFND(code,[output_error])
eg
        IF (ACDFND(23),.TRUE.) THEN failed to find Attribute Code 23
```

This routine is called to find the given code in the attribute code definition table. It sets up variables in COMMON/FRTACD/. If the optional argument "output_error" is .TRUE. (default) then an error message is output when "code" is not found in the ACD table.

If "code" is not found, the common block is filled in with default values, assuming an integer or real attribute data type, depending on the result of a call of the IFFLIB routine IS_REAL_AC(code).
The following values are set:-

```
        ACD_CODE          = -1
        ACD_NAME          = '?'
        ACD_NAMELEN      =  1

        ACD_DATA_TYPE    = 1 or 2

        ACD_MIN_MAX_xx  is set to the relevant default value
```

5.10 *ACDFND_NAME*

```
        failed = ACDFND_NAME(name,[ret])
```

where ret is an optional integer argument which receives any
error code returned by LSLLIB. If it is absent then the
corresponding error message is output; when present ACDFND_NAME
does not output any error messages.

```
eg
        IF (ACDFND_NAME(HEIGHT)) THEN failed to find HEIGHT
```

This routine is called to find the representation of a given attribute, where the name of the attribute is known.
If successful, this routine sets up the variables in COMMON/FRTACD/.
If the routine fails, then the variables in the common block are not altered.

5.11 *SRIINI*

```
        failed = SRIINI(srifile,[hwsym],[hwcir],[hwpat],[hwlns],[hwcur])
```

the optional arguments are used if hardware symbols, circles,
patterns, line styles, and curves respectively are to be used
by the plotting routines. If required, they should be set to .TRUE.
(or in the case to hwlns to the number of line styles available).

eg

```
        IF (SRIINI('LSL$FRT:SRI.SRI',.FALSE.)) THEN failed to read file
```

This routine must be called before any other SRI library routines. It opens the given file and reads it to set up the Symbol Representation Index in the FRTSRI common block.

5.12 *SRIFND*

```
        failed = SRIFND(sc)
```

eg

```
        IF (SRIFND(23)) THEN failed to find symbol 23
```

This routine is called to find the representation of a given symbol. It sets up variables in COMMON/FRTSRI/

5.13 *SRICOL*

```
        failed = SRICOL(icol)
```

eg

```
        IF (SRICOL(7)) THEN failed to set symbol colour index 7
```

This routine is called to set the colour to be used for all successive symbols (including texts). This does not affect the current polyline and fill area colour indices in use by the calling program. If the integer argument, ICOL, is negative, or if SRICOL is not called at all, then the colours specified in the FRT or SCT entries will be used in plotting texts and symbols.

5.14 *SRIPLT*

```
        failed = SRIPLT(sc,x,y,size,angle,[hwsym],[stretch])
```

hwsym is an optional logical argument (default .FALSE.) which
should be .TRUE. when a symbol from the hardware's library
is to be plotted, rather than a symbol from the SRI.

stretch is an optional real argument (default 1.0) which specifies
a stretching factor in the X direction (before rotation). It is
mainly intended for internal use within the pattern drawing
routines.

eg

```
        IF (SRIPLT(23,5.0,9.0,12.0,0.5)) THEN failed to plot symbol 23
```

This routine is called to plot a given symbol at a given X,Y position at a given size and angle.

If plotting on a 3 dimensional device, this routine should be preceded by a call to SRI_SET_Z (q.v.)

5.15 *SRISCN*

```
        failed = SRISCN(sc,xmin,xmax,ymin,ymax)
eg
        IF (SRISCN(23,XMIN,XMAX,YMIN,YMAX)) THEN failed to scan symbol 23
```

This routine is called to scan a given symbol and return its maximum extents from the defining point if plotted at size 1.0.

5.16 *SRISLW - Set Line Width*

```
        failed = SRISLW(width)
eg
        IF (SRISLW(WIDTH)) THEN failed to set width (WIDTH.LE.0.0)
```

This routine is called to inform FRTLIB of the nominal linewidth of the graphics workstation. It must be used if thick lines are to be plotted correctly. The real argument, WIDTH, is the number of width units in the FRT file to correspond to the device nominal linewidth. WIDTH must be greater than zero.

5.17 *SRIUNI - Pass Plotting Information To FRTLIB*

```
        failed = SRIUNI(units[,scl][,angle][,enluni])

        where units  - ratio of plotter units (mm) to world units
                       such that world_units * units = plotter_units

              scl    - ratio between FRT units (mm) and world units
                       such that FRT_units * scl = world_units
                       (default 1)

              angle  - angle (in radians) that coordinates have been
                       rotated by the calling program, before passing
                       them to FRTLIB

              enluni - ratio between final plotter units after any enlargement
                       of reduction (mm) and world units such that
                       world_units * enluni = final_plotter_units
                       (default same as units above)

eg
        IF (SRIUNI(UNITS)) THEN failed to set units (UNITS.LE.0.0)
```

This routine should be called after FRTINI and before any plotting is done.

The first argument is used to set the correct thickness and spacing of hatch lines in hatched areas in symbols and texts, and also to adjust the number of interpolated points per unit in circle arcs and curves unless overridden by the fourth argument. The default is appropriate for a drawing scale such that one unit corresponds to one millimetre. If, for instance, one world unit corresponds to one centimetre on the plot, then a call to SRIUNI(10.0) should be made.

The second argument is used to convert sizes specified in sheet mm in the FRT (e.g. pattern sizes) to world coordinates.

The third argument is used to ensure that substituted symbols in patterned lines are drawn at the correct orientation, and also to rotate the hatch lines in pattern filled areas.

If no enlarging or reducing of the plots is being done, then this is the same as the first argument, but otherwise it is used to adjust the tolerances for circles and curves in the final plot.

5.18 *SRI_BOUND - Return The Boundary Of A Symbol*

```
        failed = SRI_BOUND(symno,npts,xy,border,boundtype)

        INTEGER         symno           - the number number
        INTEGER         npts            - input, size of xy array
                                          output, number of points in xy
        REAL            xy(2,npts)      - output array
        REAL            border          - border as proportion of height
        INTEGER         bound_type      - type of bounding polygon
eg
        IF (SRI_BOUND(23,npts,xy,0.35,3)) THEN failed
```

This routine is used to calculate a suitable boundary around a symbol. It must be passed an array in which to return the points. Three types of boundary may be specified by argument bound_type.

1 - a box produced by calling SRISCN. This only looks at the defining points of the symbol, so might not produce accurate results for symbols which include circle arcs or curves.

2 - a box produced by going through the motions of drawing the symbol and taking the bounding box of the resulting points. This takes proper account of circle arcs and curves.

3 - same as 2, but a convex hull around the set of points is calculated. The convex hull is the shape you would get if you pulled tight a length of string around the points.

5.19 *SRI_OFFSET_POLYGON - Offset A Polygon*

```
        call SRI_OFFSET_POLYGON( maxpoints, points, npoints, dist )
        INTEGER         maxpoints       - Max. size of offset polygon
        REAL    points(2,maxpoints)     - Input polygon/offset polygon
        INTEGER         npoints         - Actual size offset polygon
        REAL            dist            - Offset distance
```

This is a convenience routine used to offset a polygon. The polygon is expected to be anti-clockwise, in which case a positive distance offsets outwards. The polygon os expected to be open.

5.20 *SRI_SET_Z - Pass Height To FRTLIB*

```
        CALL SRI_SET_Z(height)

                REAL   height
```

This routine should be called before plotting a text or symbol with TRITXT, TRIPLT or SRIPLT. Texts and symbols will then be plotted horizontally at the specified height.

It should also be called before plotting a patterned fill area with FILLGFA. The fill area will then be plotted horizontally at the specified height.

5.21 *TRIINI*

```
        failed = TRIINI(trifile)
eg
        IF (TRIINI('LSL$FRT:TRI.TRI')) THEN failed to read file
```

This routine must be called before any other TRI library routines. It opens the given file and reads it to set up the Text Representation Index in the FRTTRI common block.

5.22 *TRIFND*

```
        failed = TRIFND(charno,font)
eg
        IF (TRIFND(65,1)) THEN failed to find character 65 ('A')
```

This routine is called to find the representation of a given character. it sets up variables in COMMON/FRTTRI/

5.23 *TRIPLT*

```
        failed = TRIPLT(charno,font,x,y,size,angle[,hwtxt])
```

eg

```
        IF (TRIPLT(65,1,5.0,9.0,12.0,0.5)) THEN failed to plot character 65
```

This routine is normally called to plot a given character at a given X,Y position at a given size and angle.

If the optional hwtxt argument is .TRUE., then the text will be plotted using hardware (via routine SRIGTX), and without reference to the TRI table.

If a text graphical type has been set up in FRTCOM, and FRTSC is negative, then the character will be plotted in italic style.

The routine can also be used to find the coordinate extent of a character, instead of plotting it. Additional arguments are passed or returned via the common block TRIEXT. Operation in this mode can be switched on by setting the variable GET_EXTENT true, and the extents are returned in MIN_X_EXT, MAX_X_EXT, MIN_Y_EXT and MAX_Y_EXT. In order to maintain compatibility with previous versions of the library and to ensure that subsequent calls to TRIPLT result in plotting, GET_EXTENT should be reset to false when the required extents have been obtained. The coordinate extents can be returned in IFF units or TRI units; for the latter, a suitable call would be

```
        IF (TRIPLT(65,1,0.0,0.0,1.0,0.0)) THEN
           failed to find extent of character 65
```

If plotting on a 3 dimensional device, this routine should be preceded by a call to SRI_SET_Z (q.v.)

5.24 *TRITXT*

```
        failed = TRITXT(charstring,font,x,y,size,angle[,hwtxt])
```

eg

```
        IF (TRITXT('Rhubarb and Custard',1,5.0,9.0,12.0,0.5)) THEN failed
```

This routine is called to plot a text string at a given X,Y position at a given size and angle. It calls TRIPLT for each character, dealing with escape character sequences such as '$A', and applying variable character spacing using the widths read from the width AC entries in the TRI file.

If the optional hwtxt argument is .TRUE., then the text will be plotted using hardware (via routine SRIGTX), and without reference to the TRI table.

If hwtxt is .FALSE. (or absent), and handling of composite characters is enabled (by defining logical name LSL$COMPOSITE_CHARACTERS as 1), then any composite characters in the string (e.g. {Zcaron}) will be replaced by their first character (Z in this case).

If plotting on a 3 dimensional device, this routine should be preceded by a call to SRI_SET_Z (q.v.)

5.25 *TRISCN*

```
        failed = TRISCN(charstring,font,width[,hwtxt])
eg
        IF (TRISCN('Rhubarb and Custard',1,WIDTH)) THEN failed
```

This routine is called to scan a text string, returning the width of the string if plotted at size 1.0. It adds the widths of each character as read from the width AC entries in the TRI file, and deals with escape character sequences such as '$A'.

If the optional hwtxt argument is .TRUE., then an attempt will be made to obtain the proper width of the hardware text (via routine SRI_STRING_WIDTH), without reference to the TRI table. If SRI_STRING_WIDTH returns .TRUE., then the width will be obtained from the TRI file as usual.

If hwtxt is .FALSE. (or absent), and handling of composite characters is enabled (by defining logical name LSL$COMPOSITE_CHARACTERS as 1), then any composite characters in the string (e.g. {Zcaron}) will be replaced by their first character (Z in this case) for the purpose of calculating the width.

5.26 *TRI_BOUND*

```
        failed = TRI_BOUND(charstring,font,ncoord,xycoord,border[,hwtxt])
```

CHARACTER*(*)	charstring	- character string
INTEGER*2	font	- text font
INTEGER*4	ncoord	- passed as the max. possible number of coordinates in xycoord, returned as the actual number forming the border
REAL	xycoord(ncoord)	- x and y coordinates of border
REAL	border	- proportion of height by which boundary should be expanded beyond text limits
LOGICAL	hwtxt	- use hardware text if possible

```
eg
        IF (TRI_BOUND('Parker''s Piece',1,n,xy,0.35)) THEN failed
```

This routine is called to find the bounding coordinates of a text string, in TRI units with respect to the locating point of the first character.

If the optional hwtxt argument is .TRUE., then an attempt will be made to obtain the bounds for hardware text (via routine SRI_STRING_BOUNDS), without reference to the TRI table. If SRI_STRING_BOUNDS returns .TRUE., then the bounds will be obtained from the TRI file.

If hwtxt is .FALSE. (or absent), then it calls TRIPLT for each character, dealing with escape character sequences such as '$A', and applying variable character spacing using the widths read from the width AC entries in the TRI file. The border argument should be less than 1.0 (the maximum height in TRI units), and unexpected results may occur if it is greater than the average character width.

5.27 *PATGPL*

```
        SUBROUTINE PATGPL(ncoord,xcoord,ycoord[,zcoord])

                INTEGER ncoord          - number of points
                REAL    xcoord(ncoord)  - x coordinates
                REAL    ycoord(ncoord)  - y coordinates
                REAL    zcoord(ncoord)  - z coordinates
eg
        CALL PATGPL(27,X,Y)
```

This routine draws the line connecting the specified points in the current pattern (set by a call of PATSET (q.v.)).

It does this by making calls to SRIPLT and SRIGPL. If the optional 4th argument is present, it precedes calls to SRIPLT by a call to SRI_SET_Z.

IMPORTANT NOTE

> If PATGPL is called with the optional 4th argument, then a replacement routine SRIGPL with 4 arguments *must* be supplied. Failure to do this may cause an access violation within PATGPL.

If a substituted symbol fails to plot, then this can be detected by a call of PATERR (q.v.)

5.28 *PATSET*

```
        failed = PATSET(patno,[hwp])

eg
        if (PATSET(12)) THEN failed to find pattern 12
```

This routine sets up the pattern to be used for drawing a linear feature, after FRTCOM has been set up by a call of FRTFND. The argument passed will normally be FRTSC from FRTCOM.

The optional integer argument hwp is used to return a line-type for a hardware pattern. For a hardware pattern to be used, the hwpat argument to SRIINI must be .TRUE., the pattern itself must specify the hardware flag and must not contain substituted symbols, and the hwp argument must be present. If hwp is returned as non-zero, then it should be passed on to GKS routine GSLN (set line-type), and the line drawn using GPL - it will then be patterned using hardware.

It fails if it cannot find the pattern, in which case subsequent calls of PATGPL will draw the line as a solid line. Failure to find a symbol to substitute does not constitute a failure, but this state can be detected by a call of PATERR (q.v.)

It should be called whenever a pattern is to be started, i.e. before drawing a new feature (even with the same pattern as the last one) and after an invisible part of a feature.

5.29 *PATACT*

```
        failed = PATACT(onoff)
```

eg

```
        if (PATACT(.FALSE.)) THEN failed deactivate pattern output
```

This routine turns on or off subsequent output from PATGPL. It may be used to turn off pattern output while an invisible section of a feature is drawn. PATGPL will maintain the phase of the pattern while not producing any output.

5.30 *PATERR*

```
        SUBROUTINE PATERR(OK,SYM,PATT)

        returns LOGICAL OK(2)   - .FALSE. if major/minor symbol
                                  has been suppressed
                INTEGER SYM(2)  - number of major/minor symbol
                INTEGER PATT    - pattern number
```

eg

```
        CALL PATERR(OK,SYM,PATT)
```

This routine is called after calls to PATSET and PATGPL to find out if there have been problems finding or drawing substituted symbols

5.31 *FILLGFA*

```
        SUBROUTINE FILLGFA(ncoord,xcoord,ycoord)

                INTEGER ncoord          - number of points
                REAL    xcoord(ncoord)  - x coordinates
                REAL    ycoord(ncoord)  - y coordinates
```

eg

```
        CALL FILLGFA(14,X,Y)
```

This routine fills the area defined by the specified points with the current area fill pattern (set by a call of FILFND (q.v.)).

If plotting on a 3 dimensional device, this routine should be preceded by a call to SRI_SET_Z (q.v.)

If there is a requirement that some segments of the boundary of the area are not drawn, then routines FRT_BEGIN_FILL and FRT_END_FILL may be used to bracket a series of calls to FILLGFA. The boundary line (assuming that the fill specifies a boundary) will only connect the points in each individual call to FILLGFA. A closing line will be drawn from the last point of the last call to FILLGFA to the first point of the first. All the points from all calls to FILLGFA will be used to define the area.

5.32 *FRT_BEGIN_FILL*

```
        SUBROUTINE FRT_BEGIN_FILL

eg
        CALL FRT_BEGIN_FILL
```

Specifies the beginning of a composite area (a series of calls to FILLGFA q.v.). A matching call to FRT_END_FILL must be used after the calls to FILLGFA to cause the area to be drawn.

5.33 *FRT_END_FILL*

```
        SUBROUTINE FRT_END_FILL

eg
        CALL FRT_END_FILL
```

Specifies the end of a composite area. The area is filled.

5.34 *GRPFCT*

```
        GBITS = GRPFCT(i,grpnum)

        INTEGER         i               - element of group bitmap
        INTEGER         grpnum          - group number

eg
        GBITS = GRPFCT(2,5) gets bits for feature codes 32-63 in group 5
```

This routine is used to access the group bitmaps pointed to by the variables in FRTGRP.CMN. It returns in a 32-bit (INTEGER*4) variable the bits corresponding to 32 consecutive feature codes in a given group (group numbers are allocated starting at 1 for the first group in the FRT). Element 0 contains feature codes 0-31, 2 contains 32-63, up to 1024 which contains 32736-32767. The function can be used (with certain limitations) as a replacement for the array GRPFCT, which appeared in previous releases of the library.

5.35 *GRPFC*

```
        ingrp = GRPFC(fc,grpnum)

        INTEGER         fc              - feature code
        INTEGER         grpnum          - group number
```

eg

```
        INGRP = GRPFC(2,5) returns .TRUE. if FC 2 in in group 5
```

This logical functions returns .TRUE. if the specified feature code is in the given group.

5.36 *FRTPRIOFND*

```
        failed = FRTPRIOFND(fc)
```

eg

```
        IF (FRTPRIOFND(12)) THEN failed to set variables in common block
```

This routine is called to find the way to draw a feature with a specified feature code if using multi-pass prioritised drawing. It sets up variables in COMMON/FRTPRIO/ as follows:

- PRIO_SEL is set to the feature code that it was called with

- PRIO_NUMBER is set to the number of priority/representation pairs that were defined for PRIO_SEL. If none were defined then PRIO_NUMBER is set to 0, and the feature should be drawn using the representation for its own feature code at the default priority (PRIO_DEFAULT).

- The first PRIO_NUMBER elements of the array PRIO_PRIO contains the priorities of the representations defined for PRIO_SEL.

- The first PRIO_NUMBER elements of the array PRIO_REP contains the representations defined for PRIO_SEL.

5.37 *PRIOPRIO*

```
        exists = PRIOPRIO(priority)

        INTEGER         priority
```

eg

```
        exists = PRIOPRIO(2) returns .TRUE. if there is a priority
        record that defines a representation  at priority 2.
```

This logical functions returns .TRUE. if the specified priority is defined in the priority table.

INDEX

ACDFND, 23
ACDFND_NAME, 23

CMDLIB, 3
Common blocks, 9
CTG, 2

Feature Code, 3
Feature Code Groups, 12
FILFND, 22
FILLGFA, 31
FRT_ACDINI, 22
FRT_BEGIN_FILL, 32
FRT_END_FILL, 32
FRT_GRAPH_INIT, 20
FRTACD, 18
FRTCOM, 4, 10
FRTFGT, 22
FRTFIL, 12
FRTFND, 21
FRTGRP, 12
FRTHKS, 4
FRTINI, 20
FRTPAT, 13
FRTPRIO, 13
FRTPRIOFND, 33
FRTSCT, 14
FRTSRI, 15
FRTTRI, 16

GKS, 3
Graphical Kernel System, 3
Graphics Interface, 3
Groups, 12
GRPFC, 10, 32
GRPFCT, 10, 32

IFFLIB, 3
Initialisation, 20, 22 to 23, 27
Introduction, 2

Legenda, 2
Library, 3
LSL$LIBRARY:FRTLIB, 3
LSL$LIBRARY:LSLFRTLIB, 3
LSL_INIT, 3, 20
LSLLIB, 3, 20

PATACT, 31
PATERR, 31
PATFND, 22
PATGPL, 30
PATSET, 30
pattern, 30
Patterned lines, 13
PRIOPRIO, 33
Priority Table, 13

SCT, 14
SRI, 15
SRI_BEGIN_AREA, 9
SRI_END_AREA, 9
SRI_OFFSET_POLYGON, 26
SRI_SET_PATTERN, 9
SRI_STRING_BOUNDS, 7
SRI_STRING_WIDTH, 7
SRIBOUND, 26
SRICOL, 4, 24
SRIFND, 24
SRIGFA, 6
SRIGPL, 5
SRIGPM, 6
SRIGQA, 5
SRIGQL, 8
SRIGQM, 6
SRIGQP, 4
SRIGSA, 5
SRIGSL, 8
SRIGSM, 6
SRIGSP, 4
SRIGST, 7
SRIGTX, 7
SRIINI, 23
SRIPLT, 24
SRISCN, 25
SRISETZ, 27
SRISLW, 4, 25
SRIUNI, 4, 25
Subroutines, 3, 20
Symbol Component Table, 14
Symbol Representation IFF file (SRI), 15

Text Representation IFF file (TRI), 16
Thick, 4
TRI, 16
TRI_BOUND, 29
TRIEXT, 17
TRIFND, 27
TRIINI, 27
TRIPLT, 27

TRISCN, 29
TRITXT, 28
VIOLIB, 3