Semper 6 *Plus*

FORTRAN PROGRAMMER'S REFERENCE

Synoptics

Trademarks:

Semper is a registered trademark of *Synoptics Limited*

No part of this manual may be copied or reproduced in any form, or by any means, without prior written permission of *Synoptics Limited.*

Manual number: SEM043
Update number: 0
Printed: May 1994

Copyright © 1994: Synoptics Ltd, All Rights Reserved

Table OF CONTENTS

Chapter 1: Overview

Chapter 2: Fortran environment

Chapter 3: Fortran routine specifications

Chapter 4: File Formats

Chapter 5: Reserved names

Chapter 1

OVERVIEW

Introduction

This manual describes Semper's Fortran interface: the means by which you can access Semper's facilities from within your own Fortran software.

The topics covered in this manual are summarised below and are described in more detail in the following chapters.

Fortran environment

This chapter describes the following topics which are not adequately covered by the Fortran routine specifications contained in Chapter 3.

- Fortran INCLUDE files
- row buffers
- data representation
- logical picture table
- picture labels
- display look-up tables
- device table
- character data
- packed names
- file names
- diagnostic and error messages
- monitor output
- temporary variables
- Fortran unit numbers
- data constants

Fortran routine specifications

Detailed specifications for the following Fortran routines are to be found in this chapter:

BCLEAR	FILCLS	FSFLUS	GENHST	OPTNO	SEMMED
BCOUNT	FILDCD	FSINIT	GETFRM	PDELTA	SEMMON
BDIFF	FILDIR	FSLINE	GETRG1	PSIGMA	SEMOPN
BFILL	FILEXI	FSLIST	GETRG2	RANGE	SEMPPN
BFORM	FILMAK	FSLURW	GETRNG	SEMBEE	SEMRNG
BLOGIC	FILOPN	FSMARK	INVFT2	SEMBRK	SEMROW
BREP	FILPAT	FSOPTN	IPACK	SEMCEN	SEMSEL
BSCAN	FILSEA	FSOVRW	IVAL	SEMCMS	SEMSOP
BSET	FILSTR	FSQBQR	IVALPN	SEMCLS	SEMTEX
BHIFT	FSAMPL	FSQCHA	KLINE	SEMCON	SEMTIT
BVALUE	FSARC	FSQLIM	KPRESS	SEMDEL	SEMTPS
CFORM	FSARRO	FSQMON	KREAD	SEMDIA	SEMWAI
DATSTR	FSBAND	FSQTRA	MARSET	SEMICS	SETVAR
EVCLOS	FSBORD	FSREGN	MCTIME	SEMINP	TIMSTR
EVFLUS	FSCIRC	FSROW	MEANSD	SEMINT	UNPACK
EVOPEN	FSCTYP	FSTEXT	MKCHAR	SEMKTX	UNSETV
EVQENQ	FSCURS	FSVIEW	MKICHA	SEMLAB	VAL
EXTRCB	FSCURV	FSXWIR	NBLANK	SEMLOG	VARINT
EXTRCT	FSDRAG	FT1D	OBEYCL	SEMLU	VARSET
EXTRNN	FSELEC	FT2D	OPT	SEMLUT	WAITS
FILCIO	FSERAS				

The routine specifications are preceded by a list of routines arranged in functional groups, a set of single-line descriptions of the routines and finally a description of the format for the routine specifications.

File formats

This chapter provides detailed descriptions of the following file formats:

- READ/WRITE file format
- INPUT/OUTPUT file format

These file formats may be used to import images into Semper and possibly to export images to other software packages. They can also be used to transfer data between Semper systems running on different host computer systems.

Reserved Names

You must choose names for your routines which do not clash with any Semper routine names. This chapter lists all the Semper routine names for all platforms on which Semper currently runs.

Chapter 2

FORTRAN ENVIRONMENT

Introduction

This chapter describes in more detail the topics concerning Semper's Fortran interface which are not adequately covered by the Fortran routine specifications contained in Chapter 3. The following topics are covered:

- Fortran INCLUDE files
- row buffers
- data representation
- logical picture table
- picture labels
- display look-up tables
- device table
- character data
- packed names
- file names
- diagnostic and error messages
- monitor output
- temporary variables
- Fortran unit numbers
- data constants

Fortran INCLUDE files

You gain access from Fortran to most of the facilities in Semper by calling various Semper routines. There are, however, some facilities and data structures which can only be accessed through Fortran COMMON blocks. All of Semper's COMMON blocks are defined in a file called COMMON which you refer to by adding the following INCLUDE statement in your Fortran routine:

```
INCLUDE 'COMMON'
```

These COMMON blocks expose all of Semper's internal workings and should be accessed only when absolutely necessary. This is particularly important if you want to maximise the chances of your code working in later versions of Semper. The list of the COMMON varaibles and arrays which you are allowed to access and use is as follows:

```
REAL       RBS(1-LNEDGE:LNBUF/LNREAL+LNEDGE,NNBUF)
REAL       RB1(LNBUF/LNREAL),RB2(LNBUF/LNREAL),RB3(LNBUF/LNREAL)
REAL       RB4(LNBUF/LNREAL),RB5(LNBUF/LNREAL),RB6(LNBUF/LNREAL)
REAL       DUMFP
INTEGER    DUMINT
LOGICAL    DUMLOG
INTEGER    DEVN(NLPS),PICN(NLPS)
INTEGER    NCOLS(NLPS),NROWS(NLPS),NLAYS(NLPS)
INTEGER    CCOLN(NLPS),CROWN(NLPS),CLAYN(NLPS)
INTEGER    CLASSN(NLPS),FORMN(NLPS),PXSAM(NLPS)
REAL       GSMIN(NLPS),GSMAX(NLPS)
INTEGER    LP1,LP2,LP3
INTEGER    ERROR,IDERR,IDERR2
INTEGER*4  I4IDER
INTEGER    NLUTS,LUTLEN,LUTMAX

CHARACTER*(RECLEN) RECORD
CHARACTER*68       IDMESS
```

There is also a whole set of data constants which dimension Semper's data structures and enumerate its data types. You get access to these data constants by refering to them by their corresponding Fortran PARAMETER names. These are all defined in the file called PARAMS.

```
INCLUDE 'PARAMS'
```

The parameters which are referred to in this manual and which you may need to use are as follows:

```
INTEGER    LNBYTE,LNINT,LNINT4,LNREAL,LNCOMP
INTEGER    NNBUF,LNEDGE
INTEGER*4  LNBUF,LNSBUF
INTEGER    NDVS,NLPS
INTEGER    NCLIMA,NCLMAC,NCLFOU,NCLSPE,NCLCOR
INTEGER    NCLUND,NCLWAL,NCLPLI,NCLHIS,NCLLUT
INTEGER    NFMBYT,NFMINT,NFMFP,NFMCOM
INTEGER    MEDVM,MEDDC,MEDDS,MEDTP,MEDFL
```

```
INTEGER    LNLAB
INTEGER    LBNC1,LBNC2,LBNR1,LBNR2,LBNL1,LBNL2
INTEGER    LBCC1,LBCC2,LBCR1,LBCR2,LBCL1,LBCL2
INTEGER    LBCLAS,LBFORM,LBWP,LBPLTY
INTEGER    LBYEAR,LBMON,LBDAY,LBHOUR,LBMIN,LBSEC
INTEGER    LBNCRR,LBRR1,LBRR2,LBNCTT,LBTT1,LBTT2
INTEGER    LUTSIZ
INTEGER    FILMAX
INTEGER    NDIMES,NDIWAR,NDIERR,NDIFAT,USERR
INTEGER    RDWRTU,RDWRT2
REAL       PI,TWOPI
```

The parameters in the file ICSET define all the recognised key or character codes.

```
INCLUDE 'ICSET'
```

The following parameters are defined in ICSET:

```
INTEGER    KSPACE,KPLING,KDQUOT,KHASH,KDOLLA,KPCENT,KAMPER,KQUOTE
INTEGER    KBRA,KKET,KSTAR,KPLUS,KCOMMA,KMINUS,KDOT,KSLASH
INTEGER    KZERO,KONE,KTWO,KTHREE,KFOUR,KFIVE,KSIX,KSEVEN
INTEGER    KEIGHT,KNINE,KCOLON,KSEMIC,KABRA,KEQUAL,KAKET,KQUEST
INTEGER    KAT,KUCA,KUCB,KUCC,KUCD,KUCE,KUCF,KUCG
INTEGER    KUCH,KUCI,KUCJ,KUCK,KUCL,KUCM,KUCN,KUCO
INTEGER    KUCP,KUCQ,KUCR,KUCS,KUCT,KUCU,KUCV,KUCW
INTEGER    KUCX,KUCY,KUCZ,KSBRA,KBACKS,KSKET,KUP,KUNDER
INTEGER    KGRAVE,KLCA,KLCB,KLCC,KLCD,KLCE,KLCF,KLCG
INTEGER    KLCH,KLCI,KLCJ,KLCK,KLCL,KLCM,KLCN,KLCO
INTEGER    KLCP,KLCQ,KLCR,KLCS,KLCT,KLCU,KLCV,KLCW
INTEGER    KLCX,KLCY,KLCZ,KCBRA,KBAR,KCKET,KTILDE
INTEGER    KBRET,KBTAB,KBESC
INTEGER    KBDEL,KBKILL,KBINS,KBHOME,KBEND,KBREFL
INTEGER    KBUP,KBDOWN,KBLEFT,KBRITE
INTEGER    KBFUNC,KBFMAX
INTEGER    KMBUT,KMBMAX
INTEGER    KBNONE
```

Note that COMMON includes the contents of PARAMS and that PARAMS includes the contents of ICSET. So, including COMMON includes all three files. Be careful, when including the file called COMMON, to avoid using any names that clash with names in Semper's COMMON blocks. The best way to guard against this is to include type declarations for all your variables so that the Fortran compiler will fault any clash of names.

Row buffers

Semper requires a considerable amount of buffer space to store data when processing images. Access to images is restricted to reading and writing rows of pixels. Row buffers are required to store the rows of pixels in memory while they are being processed. Semper provides the 2D array RBS with space for NNBUF row buffers. NNBUF is guaranteed to be at least 6.

```
REAL RBS(1-LNEDGE:LNBUF/LNREAL+LNEDGE,NNBUF)
```

The array has two dimensions and both array subscripts can be safely declared with type INTEGER. The first subscript addresses pixels within a row buffer and the second subscript selects the row buffer. Having the second subscript is very useful when indexing, swapping and permuting row buffers.

For convenience and backwards compatability, six row buffer arrays RB1, RB2, RB3, RB4, RB5 and RB6 are equivalenced to RBS.

```
REAL RB1(LNBUF/LNREAL),RB2(LNBUF/LNREAL),RB3(LNBUF/LNREAL)
REAL RB4(LNBUF/LNREAL),RB5(LNBUF/LNREAL),RB6(LNBUF/LNREAL)

EQUIVALENCE (RB1,RBS(1,1)),(RB2,RBS(1,2)),(RB3,RBS(1,3))
EQUIVALENCE (RB4,RBS(1,4)),(RB5,RBS(1,5)),(RB6,RBS(1,6))
```

The row buffer arrays have been declared as REAL arrays, but there no reason why you can not use this space to store other types of data by equivalencing your own data arrays, for example

```
INCLUDE 'COMMON'

INTEGER IB1(LNBUF/LNINT)
COMPLEX CBN(LNBUF/LNCOMP)
EQUIVALENCE (IB1,RB1),(CBN,RBS(1,NNBUF)
```

Each row buffer is LNBUF bytes in length. Parameters are provided to define the size in bytes for all the Fortran data storage units: LNINT for INTEGER data, LNINT4 for INTEGER*4 data, LNREAL for REAL data and LNCOMP for COMPLEX data. Therefore each row buffer can contain LNBUF/LNINT integer elements, LNBUF/LNREAL floating-point elements and LNBUF/LNCOMP complex elements.

Sometimes it is useful to be able to index a few elements beyond the ends of a row buffer to avoid the complications that can arise when processing up to the edges of an image. The row buffers have been declared with extra space at each end sufficient to store LNEDGE floating-point values. LNEDGE is guaranteed to be at least 10. The row buffer array RBS allows you to address edge pixels directly. If you are concerned about the possible overhead of using a 2D array instead of a 1D array, you can declare your own 1D array and equivalence this to RBS.

For example,

```
      INCLUDE 'COMMON'

      REAL MYBUF(1-LNEDGE:LNBUF/LNREAL+LNEDGE)
      EQUIVALENCE (MYBUF(1),RBS(1,1))

      IF (SEMROW(1,MYBUF(1),NFMFP,1,1,LP1)) RETURN

      DO 10 I = 1-LNEDGE,0
         MYBUF(I) = 0.0
   10 CONTINUE

      DO 20 I = NCOLS(LP1)+1,NCOLS(LP1)+LNEDGE
         MYBUF(I) = 0.0
   20 CONTINUE
```

reads a row of pixels into a row buffer and then sets LNEDGE pixels at each end to zero. Because of the extra space allocated at each end of the buffer MYBUF, the example will never step off the ends of the row buffer.

As all the row buffers are contained in the single array RBS, you can declare and equivalence arrays larger than the row buffer arrays. For example,

```
      INCLUDE 'COMMON'

      INTEGER IBUFF(4,LNBUF/LNINT)
      EQUIVALENCE (IBUFF,RB1)
```

creates a single Fortran array which stores 4 integer values for each pixel.

The size of a row buffer has been restricted so that indexing up to the end of a single row buffer will not cause the array subscript to overflow the range of 16-bit integers (-32768 to 32767). When equivalencing large 1D arrays this may no longer be the case. The example above cannot suffer from this problem because the equivalenced array is two-dimensional and each subscript will never go outside the valid integer range. If, however, the array has to be one-dimensional, you will have to declare any subscript variables to be INTEGER*4. For example,

```
      INCLUDE 'COMMON'

      INTEGER*4 BUFLEN
      PARAMETER ( BUFLEN = (NNBUF*(LNBUF+2*LNEDGE*LNREAL))/LNINT )

      INTEGER BUFFER(BUFLEN)
      EQUIVALENCE (BUFFER,RBS)

      INTEGER*4 ILOOP

      DO 10 ILOOP = 1,BUFLEN
         BUFFER(ILOOP) = 0
   10 CONTINUE
```

clears the entire row buffer space to zero.

Some low-level routines also make use of row buffers for temporary workspace. When calling these routines you must be aware of their side-effect as far as the contents of the row buffers are concerned, otherwise you will be left wondering why it is that data in a row buffer has been corrupted. The following documented routines make use of row buffers:

	RBS	RB1	RB2	RB3	RB4	RB5	RB6
EXTRCB		*				*	*
EXTRCT		*				*	*
EXTRNN		*				*	*
GENHST			*	*			
GETRNG		*					
FT2D	*						
INVFT2	*						
MEANSD		*					
RANGE		*					
SEMCEN		*					
SEMDEL		*	*				
SEMOPN		*	*				
SEMRNG		*					

Data representation

Semper supports four different data forms for storing picture data:

- byte
- integer
- floating-point
- complex

These are the data forms you can request when calling routine SEMROW to access a row of picture data. You specify the data form with the corresponding Fortran parameter NFMBYT, NFMINT, NFMFP or NFMCOM. If the data form you specify differs from the stored data form for the picture (FORMN(LPN)), SEMROW converts the data for you. In this way there is no need to be directly concerned about the data form of a picture when accessing picture data (apart from the loss of precision when converting from a larger to a smaller data form).

The low-level routine for converting data between the various data forms is CFORM. CFORM can convert data between separate data arrays, or "in-place", where the source and destination arrays share the same start address. Any other kind of overlap between the source and destination arrays is likely to produce a garbled result. Normally, SEMROW calls CFORM for you, so it is not very often that you will need to call CFORM. The main reason for calling CFORM is to convert data to the stored data type before a call to SEMROW to write the data to the picture device. This prevents SEMROW from converting the data (you may want to write the same data several times, for example, when initialising the contents of a picture) and protects the contents of the data array (SEMROW may under certain circumstances convert the data in-place).

Semper also supports a bit-packed data representation where binary data is packed 8 pixels per byte. Conversion of data into and out the bit-packed representation is carried out with routine BFORM. BFORM can convert data in any of the standard data forms into the bit-packed form and back again. SEMROW does not recognise the bit-packed form, but you may still pass bit-packed data to SEMROW if you pretend it is byte or integer data and correspondingly adjust the picture dimensions to deal with the more compressed form. This approach is used by some of the binary morphology commands to provide temporary storage for bit-packed data.

Routines are provided to process 2D bit-packed arrays. Each row of data is bit-packed, with the data for each row rounded up to a multiple of 32 bits. This means that rows can be addressed through a Fortran INTEGER*4 array. Here is an example which packs the entire contents of the first layer of picture LPN into a bit-packed array:

```
      INCLUDE 'COMMON'

      INTEGER   I
      INTEGER*4 NWORDS,NSTART
      INTEGER*4 BITS( ... )

      NWORDS = (NCOLS(LPN) + 31) / 32

      NSTART = 1

      DO 10 I = 1,NROWS(LPN)
         IF (SEMROW(1,RB1,FORMN(LPN),I,1,LPN)) RETURN

         CALL BFORM(1,BITS(NSTART),RB1,FORMN(LPN),NCOLS(LPN))

         NSTART = NSTART + NWORDS
   10 CONTINUE
```

If the array BITS is a dummy array in a subroutine, it can be declared as a 2D array which means that the rows can be indexed directly:

```
      SUBROUTINE MYSUB( ... ,BITS, ... )

      INCLUDE 'COMMON'

      INTEGER   I
      INTEGER*4 BITS((NCOLS(LPN)+31)/32,*)

      DO 10 I = 1,NROWS(LPN)
         IF (SEMROW(1,RB1,FORMN(LPN),I,1,LPN)) RETURN

         CALL BFORM(1,BITS(1,I),RB1,FORMN(LPN),NCOLS(LPN))
   10 CONTINUE
```

Once the data is packed in this way, it can be examined by any of the routines BCOUNT, BDIFF, BSCAN and BVALUE and modified or processed by any of the routines BCLEAR, BFILL, BLOGIC, BREP, BSET and BSHIFT. For example,

```
CALL BCLEAR(1,NCOLS(LPN),BITS,NCOLS(LPN),NROWS(LPN))
```

clears the entire bitpacked array,

```
CALL BSHIFT(2,BITS,BITS,NCOLS(LPN),NROWS(LPN))
```

shifts the array 2 columns to the right, and

```
CALL BCOUNT(1,NCOLS(LPN),BITS,NCOLS(LPN),NROWS(LPN),NBITS)
```

counts the number of bits set in BITS, returning the count in NBITS.

Logical picture table

All access to Semper pictures is managed through the logical picture table. An entry for a particular picture is established in the table by calling routine SEMOPN. If SEMOPN is successful, it will return an integer number which is the logical picture number. You specify this number subsequently whenever you wish to refer to the picture. The logical picture number is simply in index into the logical picture table. The size of the logical picture table (and therefore the maximum value for a logical picture number) is defined by the parameter NLPS.

The information in the table directs the operation of all the routines that provide access to picture information, i.e. SEMROW, SEMCEN, SEMTIT, SEMRNG and SEMLAB. The routines for closing a picture and selecting a picture, SEMCLS and SEMSEL, also depend on this information and require the logical picture number as a subroutine argument. You may read some of the information directly from the table by means of the following arrays in COMMON:

DEVN(lpn)	Device number (1 to NDVS)
PICN(lpn)	Picture number (1 to 999)
NROWS(lpn),NCOLS(lpn),NLAYS(lpn)	Picture size
CCOLN(lpn),CROWN(lpn),CLAYN(lpn)	Picture origin
CLASSN(lpn)	Picture class
FORMN(lpn)	Data form

If the picture is a display picture, you can also get hold of the intensity scaling parameters (the intensity values that scale onto black and white on the display) and the data sampling interval (for undersampled images):

GSMIN(lpn),GSMAX(lpn)	Black and white levels
PXSAM(lpn)	Display sampling interval

You must not attempt to access these 3 arrays unless you are dealing with a display picture because the data will not otherwise be defined. To check for a display picture you must get hold of the device type for the picture by calling routine SEMMED:

```
LOGICAL SEMMED
INTEGER MEDIUM

IF (SEMMED(DEVN(lpn),MEDIUM)) RETURN

IF (MEDIUM.EQ.MEDDS) THEN

   This is a display picture ......

ENDIF
```

If the picture device is the display, the device type will be equal to the parameter MEDDS. The other valid picture device types are memory, disc and tape with corresponding parameters MEDVM, MEDDC and MEDTP for the device type. The other possible device type, MEDFL, is reserved for assignable log files.

Parameters have also been defined for all the possible class and form numbers as follows:

```
CLASSN(lpn) = NCLIMA, image
            = NCLMAC, macro
            = NCLFOU, fourier
            = NCLSPE, spectrum
            = NCLCOR, correlation
            = NCLUND, undefined
            = NCLWAL, walsh
            = NCLPLI, position list
            = NCLHIS, histogram
            = NCLLUT, look-up table

FORMN(lpn)  = NFMBYT, byte
            = NFMINT, integer
            = NFMFP,  fp
            = NFMCOM, complex
```

A valid logical picture number can only be supplied by SEMOPN. However, it is not always necessary to call SEMOPN directly because the command interpreter may be able to do this for you. The automatic opening of pictures is governed by entries such as "open(lp1,old)=from" or "open(lp2,new,lp1)=to" which appear in the syntax definitions for certain commands. If the command interpreter opens a picture automatically, it will store the logical picture number in one of the COMMON variables LP1, LP2 or LP3. The name that appears first inside the brackets of an open definition ("lp1" and "lp2" respectively in the examples above) specifies into which variable the logical picture number is stored.

Picture labels

All of the information which describes a picture is stored in a data structure called the picture label. The picture label consists of an array of 256 unsigned byte values (0 to 255). You can gain access to a picture label by calling routine SEMLAB. The picture label is passed to you as an INTEGER array and it contains the picture size, origin, class, form, write-protect flag, creation date and time, data range (if recorded), position list type (if Plist class picture) and title string.

Much of this information can be accessed by calling SEMOPN, and subsequently, by calling SEMCEN, SEMRNG and SEMTIT. After calling SEMOPN, the picture size, origin, class and form can also be obtained from the logical picture table. You will need to access the picture label to do any of the following things:

(1) Read or set/reset the write-protect flag

(2) Read or set the position list type

(3) Read the creation date and time

To access the information in a picture label, you have to call routine SEMLAB to transfer the information into a buffer array. Parameters define which elements of the array encode each piece of information contained in the picture label. Not all of the space in the picture label is used. Any unused portions are set to zero. Note that the unused portion of the picture label from LABEL(LBPLTY+1) to LABEL(LBNCTT-1) is reseved for future use.

The data in the picture label is arranged and accessed as follows:

```
LOGICAL SEMLAB
INTEGER IOP,LABEL(LNLAB),LPN
INTEGER NCOL,NROW,NCOL,CCOL,CROW,CLAY
INTEGER CLASS,FORM,WPF                          Type declarations
INTEGER YEAR,MONTH,DAY,HOUR,MINUTE,SECOND
CHARACTER*(LBRR2-LBRR1+1) RANGE
INTEGER PLTYPE
CHARACTER*(LBTT2-LBTT1+1) TITLE

IF (SEMLAB(IOP,LABEL,LPN)) RETURN               Read/write picture label

CALL SEMICS('Semper',LABEL,6)                   Header string = "Semper"

NCOL = 256*LABEL(LBNC1) + LABEL(LBNC2)
NROW = 256*LABEL(LBNR1) + LABEL(LBNR2)          Picture size
NLAY = 256*LABEL(LBNL1) + LABEL(LBNL2)

CCOL = 256*LABEL(LBCC1) + LABEL(LBCC2)
CROW = 256*LABEL(LBCR1) + LABEL(LBCR2)          Position of origin
CLAY = 256*LABEL(LBCL1) + LABEL(LBCL2)

CLASS = LABEL(LBCLAS)                           Picture class

FORM = LABEL(LBFORM)                            Data form

WPF = LABEL(LBWP)                               Write-protect flag
```

```
YEAR = LABEL(LBYEAR) + 1900
MONTH = LABEL(LBMON)                                    Creation date
DAY = LABEL(LBDAY)

HOUR = LABEL(LBHOUR)
MINUTE = LABEL(LBMIN)                                   Creation time
SECOND = LABEL(LBSEC)

CALL SEMCHS(RANGE,LABEL(LBRR1),LABEL(LBNCRR))           Read range string

PLTYPE = LABEL(LBPLTY)                                  Position list type

CALL SEMCHS(TITLE,LABEL(LBTT1),LABEL(LBNCTT))           Read title string
```

Where:

```
IOP = 1, read picture label
    = 2, write picture label

CLASS = NCLIMA, image
      = NCLMAC, macro
      = NCLFOU, fourier
      = NCLSPE, spectrum
      = NCLCOR, correlation
      = NCLUND, undefined
      = NCLWAL, walsh
      = NCLPLI, position list
      = NCLHIS, histogram
      = NCLLUT, look-up table

FORM = NFMBYT, byte
     = NFMINT, integer
     = NFMFP,  fp
     = NFMCOM, complex

WPF = 0, not write-protected
    = 1, write-protected

YEAR = 1900 to 2155
MONTH = 1 to 12
DAY = 1 to 31

HOUR = 0 to 23
MINUTE = 0 to 59
SECOND = 0 to 59

RANGE = ' ', no range recorded
      = '15.7,217.45' (for example), recorded picture range

PLTYPE = 0, picture is not a position list
       = 1, position list
       = 2, open curve
       = 3, closed curve

TITLE = ' ', no picture title
      = 'Mona Lisa' (for example), picture title
```

Display look-up tables

Display look-up tables define the mapping between actual pixel values and the colour/intensity displayed on the output monitor. The variable LUTLEN defines the number of entries in the look-up table and therefore the maximum number of discrete grey levels or colours that can be displayed at the same time. The output intensity levels stored in a look-up table can range from 0 to the value in variable LUTMAX corresponding to the minimum and maximum output intensities.

There are three types of look-up tables: monchrome, false colour and full colour. A monochrome look-up table consists of a single INTEGER array, LUTLEN in length, defining a series of grey intensities. False and full colour look-up tables consist of three such arrays, stored end-to-end, the first array defining red intensities, the second array defining green intensities and the third array defining blue intensities. This is can be illustrated as follows:

```
Monochrome look-up table:          grey intensity = LUTBUF(i)

False/full colour look-up table:   red intensity  = LUTBUF(i)
                                   green    "      = LUTBUF(i+LUTLEN)
                                   blue     "      = LUTBUF(i+2*LUTLEN)

where       i = look-up table entry number = 1 to LUTLEN
        LUTBUF = INTEGER look-up table array (0 <= LUTBUF(i) <= LUTMAX)
```

You should note that the values of LUTLEN and LUTMAX can sometimes only be determined at run time (typically on workstations where the window manager might have grabbed some of the available colours for displaying window borders, etc.). This is why they are variables and not Fortran parameters. The problem that this raises is that it is not possible to declare space for look-up table arrays using LUTLEN as Fortran cannot dynamically allocate storage space. To get round this problem, the parameter LUTSIZ is provided. It is guaranteed that LUTLEN will never exceed the value of LUTSIZ and therefore it is safe under all circumstances to declare the size of a look-up table array as 3*LUTSIZ. You can still declare a 2D look-up table array if you pass the array LUTBUF and the dimension LUTLEN to a subroutine:

```
INCLUDE 'COMMON'

INTEGER LUTBUF(3*LUTSIZ)
LOGICAL MYSUB

IF (MYSUB(LUTBUF,LUTLEN)) RETURN

........


LOGICAL FUNCTION MYSUB(LUTBUF,LUTLEN)

INTEGER LUTBUF(LUTLEN,3),LUTLEN

........
```

Regardless of the capabilities of the display device, Semper can maintain quite a number of look-up tables. The maximum number of look-up tables is stored in the variable NLUTS. The data is stored on Semper's work disc or memory and is accessed by calling routine SEMLUT. The MODE argument defines whether the look-up table is monochrome (MODE = 1), false colour (MODE = 2) or full colour (MODE = 3). A look-up table is loaded into the display hardware (or read back) by calling routine FSLURW.

Device table

The device table stores all the information necessary for the proper management of Semper's data storage devices. The size of the device table is determined by the Fortran parameter NDVS. Devices are numbered from 1 to NDVS.

Most of the information in the device table associated with a picture storage device is made available to you by the routine SEMOPN. It transfers the necessary picture parameters into the logical picture table. Access to picture data is implemented in such a way that you should not need to know the type of device in which a picture is stored. In the situations where it would be helpful to know, you can find out by calling the routine SEMMED. It returns any of the following medium types:

```
MEDIUM = MEDVM, memory
       = MEDDC, disc
       = MEDDS, display
       = MEDTP, tape
       = MEDFL, log file
```

The last medium type refers to a log file and will never be returned when you refer to a picture storage device.

Character data

The most convenient way to handle character or string data is to use Fortran CHARACTER variables and Fortran's string handling facilities. For various reasons (history, portability and convenience), Semper has its own way to encode characters which you need to be aware of.. Each character is represented by an integer value. Character strings are stored therefore as INTEGER arrays. For normal, printing characters, Semper uses the standard ASCII codes. The functions MKCHAR and MKICHA can be safely used to convert these single character codes between the CHARACTER and INTEGER forms, i.e.

```
CHARACTER CHA,MKCHAR
INTEGER   ICHA,MKICHA

CHA = MKCHAR(ICHA)
ICHA = MKICHA(CHA)
```

Subroutines SEMCHS and SEMICS perform the same conversion for strings of characters, i.e.

```
CHARACTER*(*) STRING
INTEGER       TEXT(*),N

CALL SEMCHS(STRING,TEXT,N)      Copies TEXT(i),i=1,N to STRING
CALL SEMICS(STRING,TEXT,N)      Copies STRING to TEXT(i),i=1,N
```

You will have to handle strings in INTEGER arrays if you call the routines SEMKTX, SEMTEX, SEMLAB and FSTEXT.

For the interactive input of text (from the terminal), the integer coding scheme has been extended so that the pressing of other keyboard keys can be reported, for example, cursor keys and function keys, as well as other types of events or conditions, e.g. mouse buttom press, no input data, etc. With the exception of codes for the Return, Tab and Escape keys, all the special key codes have values that lie outside the 7-bit ASCII code range (0 to 127).

Integer key codes are returned by routines KPRESS, KREAD, PSIGMA and PDELTA and by the Semper commands EVENT and INKEY.

Parameters have been provided to define all of Semper's key/character codes.

Printing ASCII codes:

	KSPACE	0	KZERO	@	KAT	P	KUCP	`	KGRAVE	p	KLCP
!	KPLING	1	KONE	A	KUCA	Q	KUCQ	a	KLCA	q	KLCQ
"	KDQUOT	2	KTWO	B	KUCB	R	KUCR	b	KLCB	r	KLCR
#	KHASH	3	KTHREE	C	KUCC	S	KUCS	c	KLCC	s	KLCS
$	KDOLLA	4	KFOUR	D	KUCD	T	KUCT	d	KLCD	t	KLCT
%	KPCENT	5	KFIVE	E	KUCE	U	KUCU	e	KLCE	u	KLCU
&	KAMPER	6	KSIX	F	KUCF	V	KUCV	f	KLCF	v	KLCV
'	KQUOTE	7	KSEVEN	G	KUCG	W	KUCW	g	KLCG	w	KLCW
(	KBRA	8	KEIGHT	H	KUCH	X	KUCX	h	KLCH	x	KLCX
)	KKET	9	KNINE	I	KUCI	Y	KUCY	i	KLCI	y	KLCY
*	KSTAR	:	KCOLON	J	KUCJ	Z	KUCZ	j	KLCJ	z	KLCZ
+	KPLUS	;	KSEMIC	K	KUCK	[	KSBRA	k	KLCK	{	KCBRA
,	KCOMMA	<	KABRA	L	KUCL	\	KBACKS	l	KLCL	|	KBAR
-	KMINUS	=	KEQUAL	M	KUCM	]	KSKET	m	KLCM	}	KCKET
.	KDOT	>	KAKET	N	KUCN	^	KUP	n	KLCN	~	KTILDE
/	KSLASH	?	KQUEST	O	KUCO	_	KUNDER	o	KLCO		

Special key codes:

```
KBTAB    Tab
KBRET    Return
KBESC    Escape

KBDEL    Backspace and delete character
KBKILL   Delete line
KBINS    Insert/replace mode
KBHOME   Start of line (home)
KBEND    End of line
KBREFL   Refresh line

KBUP     Cursor up
KBDOWN   Cursor down
KBLEFT   Cursor left
KBRITE   Cursor down

KBFUNC -> KBFMAX   Function key = KBFUNC + key number

KMBUT  -> KMBMAX   Mouse button = KMBUT + mouse button number

KBNONE   No key/button press
```

Packed names

The names of keys, options and variables are truncated to 3 characters so that the name can be packed into an INTEGER value. The ordinary Semper user never sees the packed representation: the command interpreter takes care of the necessary conversions into the packed form. At the Fortran level, however, the routines that access keys, options and variables require a packed name. The format for some error messages also requires packed names.

The packing format is not documented. Instead you should use the conversion routines IPACK and UNPACK to carry out the conversion into and out of the packed format. You can also carry out the conversion manually with the commands PACK and UNPACK but this approach means that you will have to encode packed names as unrecognisable INTEGER constants. You could code packed names as Fortran parameters, but this is still less convenient and less readable than using the function IPACK. For example,

```
      REAL    SIZE

      INTEGER NSIZE
      PARAMETER ( NSIZE = 30786 )

      SIZE = VAL(30786)

      SIZE = VAL(NSIZE)

      SIZE = VAL(IPACK('size'))
```

returns the floating-point value of the key/variable SIZE.

File names

You refer to data files (picture discs, image files, log files, etc.) by specifying the file name as a string of characters. The file name is broken down into a directory, name and extension string. Semper imposes an overall limit of 255 characters on the length of a file name. In addition to this the routines FILMAK and FILSEA impose the following limits when processing file names:

directoy	130 characters
name	80 characters
extension	40 characters

giving a maximum length for a file name of 250 characters. Note that the host operating system may impose further limits on the length of these componenets.

If the directory path is omitted, the action taken depends on whether the file is intended to be an existing file or a new file. For an existing file, a search is made for the file in each of the directories in Semper's file search path (see routine FILPAT). The current directory is always the first directory in the search path. The search path is obtained from the host operating system's search path which is usually stored in the environment variable called "path". For a new file, the file is created in the current directory (see routine FILDIR).

If the file extension is omitted, a default extension may be provided. For an existing file a search might be made first for the file name with one or more default extension before a search is made for the file without the file extension. The way in which the file is processed may depend on the file extension. For a new file, a default extension is invariably provided. These are some of the default file extensions Semper provides:

Picture disc	.dsk
Log file	.log
Run file	.run
Program source file	.spl
Program library	.plb
Help source file	.shl
Help library	.hlb
READ/WRITE image file	.dat
INPUT/OUTPUT image file	.pic

Diagnostic and error messages

All diagnostic information must be output by calling the routine SEMDIA. As well as the text string to be output, SEMDIA has an INTEGER argument which defines what kind of diagnostic message is being output. There are four classes of diagnostic messages which are defined by the following parameters:

NDIMES - informational message
NDIWAR - warning message
NDIERR - error message
NDIFAT - fatal error message

In normal circumstances, it is not necessary to call SEMDIA directly to output error messages because Semper provides a general mechanism for handling and reporting errors. The variable ERROR is set to zero by the command interpreter before a command is invoked. If an error is detected, the variable is set to the error number and the command returns to the command interpreter. If the command interpreter finds that the ERROR variable is not zero, it will construct a suitable error message and call SEMDIA to report the error. It will then terminate execution of any program, run file or string of commands and output traceback information (once again by calling SEMDIA) to indicate at which point the error occured.

All the Semper routines which can generate errors are LOGICAL functions. They will set the ERROR variable directly and flag the error by returning .TRUE. for the function value. A .TRUE. function return should cause an immediate return to the command interpreter unless:

(1) some form of error recovery has to be carried out first.

(2) the error is to be trapped and ignored, in which case you MUST immediately reset the ERROR variable to zero otherwise the command interpreter will still report the error when the command has finished.

The variables IDERR, IDERR2, I4IDER and IDMESS can be used to pass back more information to the command interpreter for constructing the error message string. The way in which these variables are used is documented in the file SEMPER.ERR. When reporting errors generated by your own commands, try to use an existing error code. For example,

```
INTEGER IPACK
INTEGER NAME

NAME = IPACK('nk')
IF (IVAL(NAME).LT.0) THEN
   ERROR = 3
   IDERR = NAME
   RETURN
ENDIF
```

causes the error message "?3: Bad value for nk". Alternatively, you can create new error messages and add these in the file SEMPER.ERR. The numbers for these error messages must lie in the range 900 to 949. The parameter USERR defines the first user-definable error number (900). Error number 900 already exists to allow you to output an arbitrary user-defined error message by setting variable IDMESS to the message string. For example,

```
ERROR = USERR
IDMESS = 'This is one of my own error messages'
RETURN
```

More complicated error messages using error numbers in the range 901 to 949 can be added to SEMPER.ERR. The command SHOW ERROR lists the information in SEMPER.ERR.

Monitor output

The monitor output stream provides a very useful diagnostic facility for use when developing Fortran routines. The MONITOR command allows you to turn monitor output on and off and to enable or disable for output any combination of the 16 monitor channels. Where the diagnostic output ends up depends on the current echo settings for the monitor output stream. These settings are controlled by the ECHO command.

Diagnostic text is output using routine SEMMON. The text and the calling routine's name are passed as Fortran character strings. A monitor channel number in the range 1 to NMCHAN is also passed to SEMMON. When monitor output is turned off, the variable MONIT is set to .FALSE. No calls should be made to SEMMON while this is the case, for example

```
      SUBROUTINE MYSUB

      INCLUDE 'COMMON'

      ......

      IF (MONIT) THEN
         IF (SEMMON('my diagnostic text','MYSUB',15)) RETURN
      ENDIF

      ......

      END
```

The following low-level routines also output diagnostic text with calls to SEMMON via monitor output channels 1 to 4:

	1	2	3	4
MEANSD	*			
RANGE	*			*
SEMCLS		*		
SEMDEL			*	
SEMOPN			*	
SEMROW		*		

Temporary variables

There are several COMMON variables which you may use as temporary workspace.

The variables DUMINT, DUMFP and DUMLOG can be used freely as spare variables. DUMLOG is particularly useful for invoking a LOGICAL function when you intend to ignore the function's return value, for example,

```
      DUMLOG = EVCLOS( )
```

This construction is necessary because statements like

```
      IF ( EVCLOS( ) ) CONTINUE
```

can result in the call to the function being omitted by some optimising Fortran compilers.

The variable RECORD can be used as a character buffer, typically for generating text strings for output to the console, for example,

```
      WRITE (RECORD,10) A,B
10    FORMAT ('Coefficients of fit: ',2G12.6)

      IF (SEMCON(RECORD)) RETURN
```

You should be aware that the variable is used extensively by other Semper commands and by the command interpreter to report errors. This will not cause any problems if you do not expect RECORD to remain unchanged between calls to your own routines. The other possible side-effect will occur if you switch on some forms of monitor output (see the MONITOR command). This causes some low-level routines, such as SEMDEL, SEMOPN, SEMROW, to overwrite the contents of RECORD.

Fortran unit numbers

Most of the time it should be possible to manage without making use of Fortran's input/output facilities. If, however, there is a situation where Fortran i/o has to be used, you are recommended to use the Fortran unit number defined by the parameters RDWRTU or RDWRT2. Both these unit numbers are safe to use as they are guaranteed not to be used by any low-level Semper routines.

Data constants

The following data constants are provided as Fortran parameters:

```
      INCLUDE 'PARAMS'

      PI    = 3.14159265359
      TWOPI = 2.0 * PI
```

Chapter 3

FORTRAN ROUTINE SPECIFICATIONS

Introduction

This chapter provides detailed specifications for the following Fortran routines:

BCLEAR	FILCLS	FSFLUS	GENHST	OPTNO	SEMMED
BCOUNT	FILDCD	FSINIT	GETFRM	PDELTA	SEMMON
BDIFF	FILDIR	FSLINE	GETRG1	PSIGMA	SEMOPN
BFILL	FILEXI	FSLIST	GETRG2	RANGE	SEMPPN
BFORM	FILMAK	FSLURW	GETRNG	SEMBEE	SEMRNG
BLOGIC	FILOPN	FSMARK	INVFT2	SEMBRK	SEMROW
BREP	FILPAT	FSOPTN	IPACK	SEMCEN	SEMSEL
BSCAN	FILSEA	FSOVRW	IVAL	SEMCMS	SEMSOP
BSET	FILSTR	FSQBQR	IVALPN	SEMCLS	SEMTEX
BHIFT	FSAMPL	FSQCHA	KLINE	SEMCON	SEMTIT
BVALUE	FSARC	FSQLIM	KPRESS	SEMDEL	SEMTPS
CFORM	FSARRO	FSQMON	KREAD	SEMDIA	SEMWAI
DATSTR	FSBAND	FSQTRA	MARSET	SEMICS	SETVAR
EVCLOS	FSBORD	FSREGN	MCTIME	SEMINP	TIMSTR
EVFLUS	FSCIRC	FSROW	MEANSD	SEMINT	UNPACK
EVOPEN	FSCTYP	FSTEXT	MKCHAR	SEMKTX	UNSETV
EVQENQ	FSCURS	FSVIEW	MKICHA	SEMLAB	VAL
EXTRCB	FSCURV	FSXWIR	NBLANK	SEMLOG	VARINT
EXTRCT	FSDRAG	FT1D	OBEYCL	SEMLU	VARSET
EXTRNN	FSELEC	FT2D	OPT	SEMLUT	WAITS
FILCIO	FSERAS				

These are the routines which may be called by user-written Fortran modules.

The routine specifications are preceded by a list of routines arranged in functional groups, a set of single-line descriptions of the routines and finally a description of the format for the routine specifications.

List of Fortran routines by functional group

Key, option and variable access

Key/variable value	IVAL	VAL	IVALPN	VARINT	
Variable table	VARSET	SETVAR	UNSETV	SEMLU	
Options	OPT	OPTNO	GETFRM		
Text keys	SEMTEX	SEMKTX	FILSTR		
Packed names	IPACK	UNPACK			

Picture access

Picture number/open/close/delete/select	SEMPPN	SEMOPN	SEMCLS	SEMDEL	SEMSEL
Row/origin/title/label/range	SEMROW	SEMCEN	SEMTIT	SEMLAB	SEMRNG

Display access

Initialisation	FSOPTN	MARSET	FSINIT		
Control	FSFLUS	FSELEC	FSVIEW		
Annotation	FSARC	FSARRO	FSBORD	FSCIRC	FSCURV
	FSERAS	FSLINE	FSLIST	FSMARK	FSTEXT
Image and overlay access	FSROW	FSOVRW			
Rubberbanding and dragging	FSBAND	FSDRAG			
Cursor control and input	FSCTYP	FSCURS	FSXWIR		
Sub-regions and sampling grid	FSREGN	FSAMPL			
Display look-up tables	FSLURW	SEMLUT			
Enquiry routines	FSQBOR	FSQCHA	FSQLIM	FSQMON	FSQTRA

File access

File open/close/access	FILOPN	FILCLS	FILCIO
File existence/search	FILEXI	FILSEA	
Decoding path names	FILDCD	FILMAK	
Current directory and file search path	FILDIR	FILPAT	

Events and wait

Initialisation and control	EVOPEN	EVCLOS	EVFLUS	EVQENQ
Keyboard and mouse buttons	KLINE	KPRESS	KREAD	
Pointer movement	PDELTA	PSIGMA		
Break processing	SEMBRK			
Timed wait	SEMWAI	WAITS		

Terminal and log file output

Logical output	SEMCON	SEMDIA	SEMINP	SEMLOG	SEMMON
Paged output	SEMSOP	SEMTPS			
Beep	SEMBEE				

Date and time

Current date and time	MCTIME	
Date and time strings	DATSTR	TIMSTR

Picture processing

Extract	EXTRCB	EXTRCT	EXTRNN
Fourier transform	FT1D	FT2D	INVFT2
Histogram	GENHST		
Survey	MEANSD	RANGE	

Copying and converting data

BFORM CFORM

Processing bit-packed data

Logic/shift	BLOGIC	BSHIFT		
Count/compare/range	BCOUNT	BDIFF	BSCAN	
Read	BVALUE			
Write	BCLEAR	BSET	BFILL	BREP

Sub-regions

GETRG1 GETRG2

Character and string handling

MKCHAR MKICHA SEMCHS SEMICS NBLANK

Device type

SEMMED

Executing commands

OBEYCL

Enquiring Semper's interactive state

SEMINT

Brief description of Fortran routines

Key, option and variable access

FILSTR	Returns the file name specified by means of the **name** key or the **again** option
GETFRM	Returns the data form specified by means of the option **byte, integer, fp** or **complex**
IVAL	Returns an integer key value
IVALPN	Returns a key value as a fully specified picture number
OPT	Checks to see if a command line option is turned on
OPTNO	Checks to see if a command line option is turned off
SEMKTX	Returns a string specified by means of a text key (prompts if the key is not present)
SEMLU	Provides general access to Semper's variable table
SETVAR	Sets the value of a Semper variable
UNSETV	Unsets a Semper variable
VAL	Returns a key value
VARINT	Checks to see if the value of a variable is an integer
VARSET	Checks to see if a Semper variable is set

Packed names

IPACK	Returns the packed integer representation for a name
UNPACK	Decodes a packed integer name

Picture access

SEMCEN	Reads/writes the picture origin
SEMCLS	Closes a picture
SEMDEL	Deletes a picture
SEMLAB	Reads/writes the picture label
SEMOPN	Opens a picture
SEMPPN	Returns a fully specified picture number
SEMRNG	Reads/writes/deletes the picture range
SEMROW	Reads/writes a row of picture data
SEMSEL	Selects the current picture
SEMTIT	Reads/writes the picture title string

Display access

FSAMPL	Returns the sampling grid parameters for a graphics sub-region
FSARC	Draws a circular arc
FSARRO	Draws an arrow
FSBAND	Draws/undraws an open or closed curve
FSBORD	Draws the current graphics border
FSCIRC	Draws a circle
FSCTYP	Defines the current cursor type
FSCURS	Switches the cursor on/off or positions the cursor
FSCURV	Draws an open or closed curve
FSDRAG	Draws/undraws an open or closed curve relative to an anchor position
FSELEC	Returns the current display parameters in variables **display, fs** and **cframe**
FSERAS	Clears a sub-region of the image and/or overlay plane
FSFLUS	Flushes any buffered graphical output to the display
FSINIT	Initialises the graphics coordinate system
FSLINE	Draws a line
FSLIST	Draws a series of points
FSLURW	Reads/writes the contents of a hardware look-up table
FSMARK	Draws a point
FSOPTN	Decodes the **picture/partition/frame** keys
FSOVRW	Reads/writes a sub-region of the overlay plane
FSQBOR	Returns the limits of the current graphics border
FSQCHA	Returns the character width and height
FSQLIM	Returns the current graphics clipping limits
FSQMON	Returns the current frame, zoom factor and monitor limits
FSQTRA	Returns the parameters for the transformation from graphics to display coordinates
FSREGN	Returns the limits of a graphics sub-region
FSROW	Reads/writes a subregion of the image plane
FSTEXT	Draws a string of characters
FSVIEW	Make the current graphics area visible
FSXWIR	Returns a cursor-defined graphics position
MARSET	Decodes the **mark** key
SEMLUT	Reads/writes the contents of a Semper look-up table

File access

FILCIO	Reads/writes a string of characters as a record in a file
FILCLS	Closes a file
FILDCO	Splits a file name into directory, name and extension
FILDIR	Returns the pathname of the current directory
FILEXI	Checks for the existence of a file
FILMAK	Combines directory, name and extension into a file name
FILOPN	Opens a file
FILPAT	Returns a directory pathname in the file search path
FILSEA	Searches for a file

Events and wait

EVCLOS	Restores the state of the event queues
EVFLUS	Flushes the contents of the event queues
EVOPEN	Enables/disables event queues
EVQENQ	Returns the current state of the event queues
KLINE	Prompts for and reads a string of characters from the keyboard
KPRESS	Waits for and reads a key or mouse button press
KREAD	Reads a key or mouse button press
PDELTA	Reads all pointer movements
PSIGMA	Sums all pointer movements
SEMBRK	Checks for any break events
SEMWAI	Waits for a specified time or a break event
WAITS	Waits for a specified time

Terminal and log file output

SEMBEE	Makes the terminal beep
SEMCON	Writes a line of text to the console output stream
SEMDIA	Writes a line of text to the diagnostic output stream
SEMINP	Writes a line of text to the output stream devoted to the input of non-command text
SEMLOG	Writes a line of text to the log output stream
SEMMON	Writes a line of text to the monitor output stream
SEMSOP	Signals the start of a new page of output text
SEMTPS	Returns the current terminal page size

Date and time

DATSTR	Formats data values into a string
MCTIME	Returns the current date and time
TIMSTR	Formats time values into a string

Picture processing

EXTRCB	Extracts picture data using bi-cubic interpolation
EXTRCT	Extracts picture data using bi-linear interpolation
EXTRNN	Extracts picture data using nearest neighbour sampling
FT1D	Calculates the Fourier transform of a 1D array of data
FT2D	Calculates the 2D Fourier transform of a picture
GENHST	Generates the histogram of a picture
GETRNG	Returns the intensity range of a picture
INVFT2	Calculates the inverse 2D Fourier transform of a picture
MEANSD	Returns the intensity range, mean and standard deviation of a picture
RANGE	Provides general access to the range information of a picture

Copying and converting data

BFORM	Converts data to/from the bit-packed representation
CFORM	Converts data between the byte, integer, floating-point and complex data forms

Processing bit-packed data

BCLEAR	Clears columns of pixels
BCOUNT	Counts pixels
BDIFF	Compares arrays of pixels
BFILL	Fills columns of pixels with a specified value
BLOGIC	Carries out a specified logical operation between arrays of pixels
BREP	Replicates columns of pixels
BSCAN	Returns the range of an array of pixels
BSET	Set columns of pixels
BSHIFT	Shifts an array of pixels to the left or right
BVALUE	Returns the value of a specified pixel

Sub-regions

GETRG1	Returns a rectangular sub-region defined by means of the sub-region keys and options
GETRG2	Returns an angled/skewed sub-region defined by means of the sub-region keys and options

Character and string handling

MKCHAR	Converts an integer ASCII code to a single Fortran character
MKICHA	Converts a single Fortran character to an integer ASCII code
NBLANK	Returns the non-blank length of a Fortran character string
SEMCHS	Converts an array of integer ASCII codes to a Fortran character string
SEMICS	Converts a Fortran character string to an array of integer ASCII codes

Device type

SEMMED	Returns the medium type of a device

Executing commands

OBEYCL	Queues a string of Semper commands for execution on return to the command

Enquiring Semper's interactive state

SEMINT	Returns information about Semper's interactive state

Specification format

The Fortran routine specifications have the following format:

```
<routine type> <routine name>([<argument name>[,<argument name>[, ....]]])

<brief description>

Argument          Type              I/O

<argument name>   <argument type>   <i/o status>   <argument description>

 ..............    ..............    ..........    ....................

[<COMMON variables>]

[<row buffers>]

<detailed description>
```

Here, for example, is the specification for the routine FSLINE:

```
LOGICAL FUNCTION FSLINE(X1,Y1,X2,Y2)

Draws a line from (X1,Y1) to (X2,Y2) in the overlay plane.  FSINIT must be
called first to establish the graphics coordinate system.

Argument    Type      I/O

X1          R         I      start point of the line in graphics coordinates
Y1          R         I      .

X2          R         I      end point of the line in graphics coordinates
Y2          R         I      .

Lines are truncated at the graphics limits (see FSQLIM).

FSLINE returns .FALSE. when successful.
```

The field `<routine type>` specifies the data type of any function return value.

```
<routine type> = SUBROUTINE (no return value)
               = LOGICAL FUNCTION
               = INTEGER FUNCTION
               = REAL FUNCTION
               = CHARACTER*n FUNCTION
```

The field <argument type> specifies whether an argument is a variable, array or character string.

```
<argument type> = <variable>
                = <array>
                = <character string>
```

Variables may have the following data types:

```
<variable> = L          variable type = LOGICAL
           = I                        = INTEGER
           = I*4                      = INTEGER*4
           = R                        = REAL
```

Arrays have a data type and one or more dimensions.

```
<array> = <array type>(<array dimension>[,<array dimension>])

<array type> = B      array type = byte (not a standard Fortran data type)
             = I                 = INTEGER
             = I*4               = INTEGER*4
             = R                 = REAL
             = CP                = COMPLEX

<array dimension> = 7, LNLAB, etc.   constant size array
                  = <parameter>      array size is determined at run time
                  = *                variable size array
```

Character strings are all declared with variable size.

```
<character string> = C*(*)
```

The <i/o status> field specifies whether an argument is read or written to (or both). If an argument is input only, its value will not be changed by the routine. Constants may only be passed as input only arguments.

```
<i/o status> = I     parameter is input only (possibly a constant)
             =   O   parameter is output only
             = I/O   parameter is input/output (read and changed)
```

Where the i/o status of an argument depends on another argument, this is specified in the corresponding <argument description>. For example,

```
Argument    Type      I/O

IOP         I         I      IOP = 1, read data
                                 = 2, write data

DATA        I(*)        O    if IOP = 1, input data array
                        I              = 2, output data array
```

A small number of routines communicate information by means of Fortran COMMON variables. If so, the variables are described in a <COMMON variables> section with a format similar to that used to describe routine arguments.

```
COMMON variables
----------------

Variable          Type              I/O

<variable name>  <variable type>  <i/o status>  <variable description>

 ............     ............     .........     ..................
```

Some routines use one or more of Semper's row buffer arrays RB1 to RB6 or the combined row buffer array RBS. This use is flagged in a <row buffers> section with a format similar to that used to describe routine arguments.

```
ROW buffers
-----------

Buffer          Type              I/O

<buffer name>  <buffer type>     <i/o status>      <buffer description>

 ...........    ...........       .........        ..................
```

Row buffers are used mainly as workspace but a few routines also use them to pass arrays of data. The <buffer description> will explain how the corresponding row buffer is used.

The final section <detailed description> will provide more details about the operation of a routine where it is needed, including the following:

- Background information
- Function return values
- Warnings about possible side effects, such as use of row buffers
- Algorithms and formulae
- Cross references to related routines

```
SUBROUTINE BCLEAR(IBIT1,IBIT2,BITS,NBIT,NROW)
```

Clears bits IBIT1 to IBIT2 in all the rows of bit array BITS.

Argument	Type	I/O	
IBIT1	I	I	start and end bit position
IBIT2	I	I	.
BITS	I*4(*)	I/O	bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BCLEAR assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if IBIT1 or IBIT2 is outside the range 1 to NBIT, or if IBIT2 is less than IBIT1, BCLEAR does nothing.

See also BFORM, BSET and BFILL.

```
SUBROUTINE BCOUNT(IBIT1,IBIT2,BITS,NBIT,NROW,NBITS)
```

Scans bits IBIT1 to IBIT2 in all the rows of bit array BITS and returns the number of set (non-zero) bits in NBITS.

Argument	Type	I/O	
IBIT1	I	I	start and end bit position
IBIT2	I	I	.
BITS	I*4(*)	I	bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows
NBITS	I*4	O	number of set (non-zero) bits

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BCOUNT assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if IBIT1 or IBIT2 is outside the range 1 to NBIT, or if IBIT2 is less than IBIT1, BCOUNT sets NBITS to -1.

See also BFORM and BSCAN.

```
LOGICAL FUNCTION BDIFF(IBIT1,IBIT2,BITS1,BITS2,NBIT,NROW)
```

Compares bits IBIT1 to IBIT2 in all the rows of bit arrays BITS1 and BITS2 and returns .TRUE. if any bits differ.

Argument	Type	I/O	
IBIT1	I	I	start and end bit position
IBIT2	I	I	.
BITS1	I*4(*)	I	source bit array
BITS2	I*4(*)	I	source bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BDIFF assumes the following dimensions for BITS1 and BITS2:

```
INTEGER*4 BITS1(NWORD,NROW),BITS2(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if IBIT1 or IBIT2 is outside the range 1 to NBIT, or if IBIT2 is less than IBIT1, .FALSE. is returned.

See also BFORM and BSCAN.

```
SUBROUTINE BFILL(BIT,IBIT1,IBIT2,BITS,NBIT,NROW)
```

Sets bits IBIT1 to IBIT2 in all the rows of bit array BITS to 0 or 1 according to whether BIT is respectively zero or non-zero.

Argument	Type	I/O	
BIT	I	I	BIT = 0, clear bits ~= 0, set bits
IBIT1	I	I	start and end bit position
IBIT2	I	I	.
BITS	I*4(*)	I/O	bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BFILL assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if IBIT1 or IBIT2 is outside the range 1 to NBIT, or if IBIT2 is less than IBIT1, BFILL does nothing.

BFILL is implemented in terms of the routines BCLEAR and BSET:

```
IF (BIT.EQ.0) THEN
   CALL BCLEAR(IBIT1,IBIT2,BITS,NBIT,NROW)
ELSE
   CALL BSET(IBIT1,IBIT2,BITS,NBIT,NROW)
ENDIF
```

See also BFORM, BCLEAR and BSET.

```
SUBROUTINE BFORM(IOP,BITS,DATA,FORM,N)
```

Converts data into (IOP = 1) or out of (IOP = 2) the bit-packed representation in bit array BITS. Unpacked data in the array DATA can take any of the forms supported by the form conversion routine CFORM: byte, integer, floating-point or complex data. FORM specifies the data form and N specifies the number of data values to convert.

Argument	Type	I/O	
IOP	I	I	IOP = 1, convert data of given form to bit-packed form = 2, convert bit-packed data to data of given form
BITS	I*4(*)	O	if IOP = 1, output bit array
		I	= 2, input bit array
DATA		I	if IOP = 1, input data array
		O	= 2, output data array
	B(*)		if FORM = NFMBYT, byte data array
	I(*)		= NFMINT, integer data array
	R(*)		= NFMFP, floating-point data array
	CP(*)		= NFMCOM, complex data array
FORM	I	I	form in which data is stored in data array DATA FORM = NFMBYT, byte data = NFMINT, integer data = NFMFP, floating-point data = NFMCOM, complex data
N	I	I	number of data values to convert

BFORM can convert data in-place (where BITS and DATA point to the same address in memory).

The array of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the array is 32-bit aligned. This means that BFORM assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of the array are ignored.

If N is less than 1, or if IOP is not equal to 1 or 2, or if FORM is not equal to NFMBYT, NFMINT, NFMFP or NFMCOM, BFORM does nothing.

See BCLEAR, BDIFF, BFILL, BLOGIC, BREP, BSCAN, BSET, BSHIFT and BVALUE for details of how to process bit-packed data.

```
SUBROUTINE BLOGIC(IOP,SBITS,DBITS,NBIT,NROW)
```

Carries out the binary logical operation specified by IOP between bit arrays SBITS and DBITS, leaving the result in DBITS.

Argument	Type	I/O	
IOP	I	I	IOP = 0, DBITS = all zeros = 1, DBITS = and(not(SBITS),not(DBITS)) = not(or(SBITS,DBITS)) = 2, DBITS = and(not(SBITS),DBITS) = 3, DBITS = not(SBITS) = 4, DBITS = and(SBITS,not(DBITS)) = 5, DBITS = not(DBITS) = 6, DBITS = xor(SBITS,DBITS) = xor(not(SBITS),not(DBITS)) = 7, DBITS = or(not(SBITS),not(DBITS)) = not(and(SBITS,DBITS)) = 8, DBITS = and(SBITS,DBITS) = 9, DBITS = xor(SBITS,not(DBITS)) = xor(not(SBITS),DBITS) = 10, DBITS = DBITS = 11, DBITS = or(not(SBITS),DBITS) = 12, DBITS = SBITS = 13, DBITS = or(SBITS,not(DBITS)) = 14, DBITS = or(SBITS,DBITS) = 15, DBITS = all ones
SBITS	I*4(*)	I	first source bit array
DBITS	I*4(*)	I	second source bit array
		O	destination bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

BLOGIC supports all the 16 possible logical operations between binary operands including the basic ones such as OR (IOP = 14), AND (IOP = 4), XOR (IOP = 6), NOT (IOP = 3 or 5), copy (IOP = 12), clear (IOP = 0) and set (IOP = 15). The operation is applied to corresponding bits in the source arrays SBITS and DBITS with the result stored in DBITS. SBITS and DBITS are allowed to point to the same address in memory.

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BLOGIC assumes the following dimensions for SBITS and DBITS:

```
INTEGER*4 SBITS(NWORD,NROW),DBITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if IOP is outside the range 0 to 15, BLOGIC does nothing.

See also BFORM and BSHIFT.

```
SUBROUTINE BREP(IBIT,IBIT1,IBIT2,BITS,NBIT,NROW)
```

Sets bits IBIT1 to IBIT2 in each row of bit array BITS to the value of the bit at position IBIT in the same row.

Argument	Type	I/O	
IBIT	I	I	position of bit to replicate
IBIT1	I	I	start and end bit position
IBIT2	I	I	.
BITS	I*4(*)	I/O	bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BREP assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if IBIT, IBIT1 or IBIT2 is outside the range 1 to NBIT, or if IBIT2 is less than IBIT1, BREP does nothing.

See also BFORM.

```
SUBROUTINE BSCAN(IBIT1,IBIT2,BITS,NBIT,NROW,BMIN,BMAX)
```

Scans bits IBIT1 to IBIT2 in all the rows of bit array BITS and returns their range in BMIN and BMAX.

Argument	Type	I/O	
IBIT1	I	I	start and end bit position
IBIT2	I	I	.
BITS	I*4(*)	I	bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows
BMIN	I	O	minimum value of scanned bits
BMAX	I	O	maximum value of scanned bits

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BSCAN assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

BSCAN always sets BMIN and BMAX to 0 or 1. Normally, BMIN < BMAX, but if NBIT or NROW is less than 1, or if IBIT1 or IBIT2 is outside the range 1 to NBIT, or if IBIT2 is less than IBIT1, BSCAN sets BMIN to 1 and BMAX to 0.

See also BFORM and BDIFF.

```
SUBROUTINE BSET(IBIT1,IBIT2,BITS,NBIT,NROW)
```

Sets bits IBIT1 to IBIT2 in all the rows of bit array BITS.

Argument	Type	I/O	
IBIT1	I	I	start and end bit position
IBIT2	I	I	.
BITS	I*4(*)	I/O	bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BSET assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if IBIT1 or IBIT2 is outside the range 1 to NBIT, or if IBIT2 is less than IBIT1, BSET does nothing.

See also BFORM, BCLEAR and BFILL.

SUBROUTINE BSHIFT(RSHFT,SBITS,DBITS,NBIT,NROW)

Shifts all the rows of bits in bit array SBITS by RSHFT positions to the right, leaving the result in DBITS.

Argument	Type	I/O	
RSHFT	I	I	number of bit positions for shift operation RSHFT < 0, left shift = 0, copy SBITS to DBITS > 0, right shift
SBITS	I*4(*)	I	source bit array
DBITS	I*4(*)	O	destination bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

BSHIFT supports left shifts (RSHFT < 0) and right shifts (RSHFT > 0). If RSHFT is zero, BSHIFT copies the contents of SBITS to DBITS. Bits shifted out of a row are discarded and bits shifted in are undefined. BSHIFT is able to shift data in-place (where SBITS and DBITS point to the same address in memory).

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BSHIFT assumes the following dimensions for SBITS and DBITS:

```
INTEGER*4 SBITS(NWORD,NROW),DBITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

If NBIT or NROW is less than 1, or if ABS(RSHFT) is greater than or equal NBIT, BSHIFT does nothing.

See also BFORM and BLOGIC.

```
INTEGER FUNCTION BVALUE(IBIT,JBIT,BITS,NBIT,NROW)
```

Returns the value of the bit at position (IBIT,JBIT) in bit array BITS.

Argument	Type	I/O	
IBIT	I	I	column and row position of bit
JBIT	I	I	.
BITS	I*4(*)	I	bit array
NBIT	I	I	number of bits in each row
NROW	I	I	number of rows

Each row of bit-packed pixels is rounded up to the next 32-bit (INTEGER*4) word to ensure that the start of every row is 32-bit aligned. This means that BVALUE assumes the following dimensions for BITS:

```
INTEGER*4 BITS(NWORD,NROW)
```

where NWORD = (NBIT + 31)/32. The extra (NWORD*32 - NBIT) bits at the end of each row are ignored.

BVALUE always returns a value of 0 or 1. If NBIT or NROW is less than 1, or if IBIT is outside the range 1 to NBIT, or if JBIT is outside the range 1 to NROW, 0 is returned.

See also BFORM, BCLEAR, BSET and BFILL.

```
SUBROUTINE CFORM(INBUF,OUTBUF,INFORM,OUTFRM,N)

Converts data between two specified forms.

Argument    Type      I/O

INBUF                 I     input data array

            B(*)            if INFORM = NFMBYT, byte data array
            I(*)                      = NFMINT, integer data array
            R(*)                      = NFMFP,  floating-point data array
            CP(*)                     = NFMCOM, complex data array

OUTBUF                  O   output data array

            B(*)            if OUTFRM = NFMBYT, byte data array
            I(*)                      = NFMINT, integer data array
            R(*)                      = NFMFP,  floating-point data array
            CP(*)                     = NFMCOM, complex data array

INFORM      I         I     form in which data are read from INBUF

                            INFORM = NFMBYT, byte
                                   = NFMINT, integer
                                   = NFMFP,  floating-point
                                   = NFMCOM, complex

OUTFRM      I         I     form in which data are to written to OUTBUF

                            OUTFRM = NFMBYT, byte
                                   = NFMINT, integer
                                   = NFMFP,  floating-point
                                   = NFMCOM, complex

N           I*4       I     number of data values to convert
```

CFORM is the routine used by Semper to perform data conversion between the various data forms (CFORM is called indirectly by SEMROW when picture rows with the wrong data form are accessed). CFORM can convert data in-place (where INBUF and OUTBUF point to the same address in memory). Complex data is converted to other forms by discarding the imaginary data component. Byte data is limited to the range 0 to 255.

The conversion of data that lie outside the valid range of the output form, OUTFRM, is not defined. The result will depend on the particular implementation of CFORM (for example, integer 257 may be truncated to the byte value 255 or wrapped around to byte value 1, and the floating point value 32769.0 may generate an integer overflow error when converting to integer form or be truncated to the maximum 16-bit integer 32767.)

Since Fortran does not in general support the byte data type, byte data arrays will need to be declared with one of the available Fortran data types, for example INTEGER or REAL.

See also BFORM and SEMROW.

```
CHARACTER*11 FUNCTION DATSTR(IDAY,IMONTH,IYEAR)
```

Returns an 11 character string containing the specified date in text form. If any argument is out of range, asterisks '*' are substituted for it in the return string.

Argument	Type	I/O	
IDAY	I	I	day, in range 1 to 7
IMONTH	I	I	month, in range 1 to 12
IYEAR	I	I	year, in range 1000 to 9999

Output is of the form:

```
DATSTR = '28-Jun-1990'
```

See also MCTIME and TIMSTR.

LOGICAL FUNCTION EVCLOS( )

Restores the state of event queues.

EVCLOS returns .FALSE. unless an error has been detected.

Restores the event queues to the state last recorded by EVOPEN, i.e. the state prior to the first call to EVOPEN after the last call to EVCLOS, or, if EVOPEN has not been called, the initial state at the beginning of the SEMPER session.

This arrangement means that EVOPEN can be called any number of times, activating and deactivating various queues, before the terminating call to EVCLOS.

See also EVOPEN, KREAD, PSIGMA and PDELTA.

```
LOGICAL FUNCTION EVFLUS(LKEY,LBUT,LPOINT)
```

Flushes specified event queues, in preparation for call(s) to other event routines.

Argument	Type	I/O	
LKEY	L	I	LKEY = .TRUE., flush keyboard queue
LBUT	L	I	LBUT = .TRUE., flush mouse button queue
LPOINT	L	I	LPOINT = .TRUE., flush pointer queue

EVFLUS returns .FALSE. unless an error is detected

See also EVOPEN, EVCLOS and EVQENQ.

```
LOGICAL FUNCTION EVOPEN(LKEY,LBUT,LPOINT)
```

Activates or deactivates specified queues, in preparation for call(s) to other event routines. May also record the current state of queues for restoration by EVCLOS.

Argument	Type	I/O	
LKEY	L	I	LKEY = .TRUE., enable keyboard events = .FALSE., disable " "
LBUT	L	I	LBUT = .TRUE., enable mouse button events = .FALSE., disable " " "
LPOINT	L	I	LPOINT = .TRUE., enable pointer events = .FALSE., disable " "

The first time EVOPEN is called after a call to EVCLOS (or the first time it is called in a SEMPER session), EVOPEN records the current state of the event queues. This state can then be restored by a call to EVCLOS. This arrangement means that EVOPEN can be called any number of times, activating and deactivating various queues, before the terminating call to EVCLOS.

EVOPEN returns .FALSE. unless an error has been detected.

See also EVCLOS, EVFLUS, EVQENQ, KREAD, PSIGMA and PDELTA.

```
LOGICAL FUNCTION EVQENQ(LKEY,LBUT,LPOINT)
```

Returns which of the keyboard, button and pointer event queues are active.

Argument	Type	I/O	
LKEY	L	O	LKEY = .TRUE., keyboard queue is active = .FALSE., queue is inactive (dormant)
LBUT	L	O	LBUT = .TRUE., mouse button queue is active = .FALSE., queue is inactive (dormant)
LPOINT	L	O	LPOINT = .TRUE., pointer queue is active = .FALSE., queue is inactive (dormant)

EVQENQ returns .FALSE. unless an error is detected.

See also EVOPEN, EVCLOS and EVFLUS.

```
LOGICAL FUNCTION EXTRCB(LPN,N,FORM,LAYER,LBLANK,VALUE)
```

EXTRCB performs general extraction from the the specified layer of the source picture with logical picture number LPN, using bi-cubic interpolation. The positions of the pixels must be supplied in the row buffers RB3 (for X) and RB4 (for Y) as floating point column and row positions. The results are returned in RB2.

Argument	Type	I/O	
LPN	I	I	logical picture number of source picture
N	I	I	number of pixel values to extract
FORM	I	I	data form for the result FORM = NFMBYT, byte = NFMINT, integer = NFMFP, floating-point = NFMCOM, complex
LAYER	I	I	source picture layer number
LBLANK	L	I	LBLANK = .TRUE., points outside the picture are returned as having the value specified by VALUE = .FALSE., positions will 'wrap-around' the picture boundaries
VALUE	R(2)	I	background value VALUE(1) = real component of background value (ignored if LBLANK is .FALSE.) VALUE(2) = imaginary component of background value (ignored if LBLANK is .FALSE. or complex result (FORM = NFMCOM) not required)

Row buffers

Buffer	Type	I/O	
RB3	R(N)	I	array of floating-point column positions
RB4	R(N)	I	array of floating-point row positions
RB2	B(N)	O	if FORM = NFMBYT
	R(N)	O	= NFMFP
	I(N)	O	= NFMINT
	CP(N)	O	= NFMCOM
RB1			used as workspace
RB5			used as workspace
RB6			used as workspace

The sampling positions supplied in RB3 and RB4 are floating-point column and row positions. Positions in picture coordinates are converted to column and row positions in the following way:

```
XCOL = REAL(CCOLN(LPN)) + XPIC
YROW = REAL(CROWN(LPN)) - YPIC
```

If complex results are required, the data are extracted directly in complex form. For results in any of the other forms, the data are extracted in floating-point form and if necessary converted to byte or integer form.

WARNING: Row buffers RB1, RB5, and RB6 are used as workspace. Values held in these buffers will not be preserved.

EXTRCB returns .FALSE. unless an error is detected (e.g. source picture has too many rows to process, bad layer number).

See also EXTRCT and EXTRNN.

```
LOGICAL FUNCTION EXTRCT(LPN,N,FORM,LAYER,LBLANK,VALUE)
```

EXTRCT performs general extraction from the the specified layer of the source picture with logical picture number LPN, using bi-linear interpolation. The positions of the pixels must be supplied in the row buffers RB3 (for X) and RB4 (for Y) as floating point column and row positions. The results are returned in RB2.

Argument	Type	I/O	
LPN	I	I	logical picture number of source picture
N	I	I	number of pixel values to extract
FORM	I	I	data form for the result FORM = NFMBYT, byte = NFMINT, integer = NFMFP, floating-point = NFMCOM, complex
LAYER	I	I	source picture layer number
LBLANK	L	I	LBLANK = .TRUE., points outside the picture are returned as having the value specified by VALUE = .FALSE., positions will 'wrap-around' the picture boundaries
VALUE	R(2)	I	background value VALUE(1) = real component of background value (ignored if LBLANK is .FALSE.) VALUE(2) = imaginary component of background value (ignored if LBLANK is .FALSE. or complex result (FORM = NFMCOM) not required)

Row buffers

Buffer	Type	I/O	
RB3	R(N)	I	array of floating-point column positions
RB4	R(N)	I	array of floating-point row positions
RB2	B(N)	O	if FORM = NFMBYT
	R(N)	O	= NFMFP
	I(N)	O	= NFMINT
	CP(N)	O	= NFMCOM
RB1			used as workspace
RB5			used as workspace
RB6			used as workspace

The sampling positions supplied in RB3 and RB4 are floating-point column and row positions. Positions in picture coordinates are converted to column and row positions in the following way:

```
XCOL = REAL(CCOLN(LPN)) + XPIC
YROW = REAL(CROWN(LPN)) - YPIC
```

If complex results are required, the data are extracted directly in complex form. For results in any of the other forms, the data are extracted in floating-point form and if necessary converted to byte or integer form.

WARNING: Row buffers RB1, RB5, and RB6 are used as workspace. Values held in these buffers will not be preserved.

EXTRCT returns .FALSE. unless an error is detected (e.g. source picture has too many rows to process, bad layer number).

See also EXTRCB and EXTRNN.

```
LOGICAL FUNCTION EXTRNN(LPN,N,FORM,LAYER,LBLANK,VALUE)
```

EXTRNN performs general extraction from the the specified layer of the source picture with logical picture number LPN, using nearest-neighbour extraction. The positions of the pixels must be supplied in the row buffers RB3 (for X) and RB4 (for Y) as floating point column and row positions. The results are returned in RB2.

Argument	Type	I/O	
LPN	I	I	logical picture number of source picture
N	I	I	number of pixel values to extract
FORM	I	I	data form for the result FORM = NFMBYT, byte = NFMINT, integer = NFMFP, floating-point = NFMCOM, complex
LAYER	I	I	source picture layer number
LBLANK	L	I	LBLANK = .TRUE., points outside the picture are returned as having the value specified by VALUE = .FALSE., positions will 'wrap-around' the picture boundaries
VALUE	R(2)	I	background value VALUE(1) = real component of background value (ignored if LBLANK is .FALSE.) VALUE(2) = imaginary component of background value (ignored if LBLANK is .FALSE. or complex result (FORM = NFMCOM) not required)

Row buffers

Buffer	Type	I/O	
RB3	R(N)	I	array of floating-point column positions
RB4	R(N)	I	array of floating-point row positions
RB2	B(N)	O	if FORM = NFMBYT
	R(N)	O	= NFMFP
	I(N)	O	= NFMINT
	CP(N)	O	= NFMCOM
RB1			used as workspace
RB5			used as workspace
RB6			used as workspace

The sampling positions supplied in RB3 and RB4 are floating-point column and row positions. Positions in picture coordinates are converted to column and row positions in the following way:

```
XCOL = REAL(CCOLN(LPN)) + XPIC
YROW = REAL(CROWN(LPN)) - YPIC
```

If integer, floating-point or complex results are required, the data are extracted directly in the required form. If byte results are required, the data are extracted in integer form and converted to byte form.

WARNING: Row buffers RB1, RB5, and RB6 are used as workspace. Values held in these buffers will not be preserved.

EXTRNN returns .FALSE. unless an error is detected (e.g. source picture has too many rows to process, bad layer number).

See also EXTRCT and EXTRCB.

```
LOGICAL FUNCTION FILCIO(HANDLE,IOP,TEXT,N)

Reads from/writes to a file specified by its file handle.

Argument      Type  I/O

HANDLE        I     I     file handle as returned by FILOPN

IOP           I     I     IOP = 1, read
                          = 2, write

TEXT          C*(*)   O   if IOP = 1, input buffer
                      I           = 2, output buffer

N             I       O   if IOP = 1, length of input record
                                      N = -1, end-of-file detected

                      I           = 2, number of characters to be written
                                      N = 0, blank record written
```

TEXT should be long enough to accommodate the input line, otherwise input will be truncated. If truncation occurs and the implementation of FILCIO allows it, N will be returned as the true length of the input record. If an attempt is made to read beyond the end of file, N is returned as -1 with TEXT as a blank string.

FILCIO returns .FALSE. unless an error occurs.

See also FILOPN and FILCLS.

```
LOGICAL FUNCTION FILCLS(HANDLE,DELETE)
```

Closes (and optionally deletes) the specified file, given the file handle returned by FILOPN.

Argument	Type	I/O	
HANDLE	I	I	file handle as returned by FILOPN
DELETE	L	I	DELETE = .TRUE. if the file is to be deleted = .FALSE. if the file is to be closed

FILCLS returns .TRUE. only in case of error.

FILCLS will close the file, if possible, even if called with the COMMON variable ERROR set to a non-zero value.

See also FILOPN.

```
LOGICAL FUNCTION FILDCD(FULNAM,PREFIX,NAME,EXTEN)
```

Decodes file name FULNAM into its prefix, name and extension components.

Argument	Type	I/O	
FULNAM	C*(*)	I	fully specified file name
PREFIX	C*(*)	O	file prefix PREFIX = ' ', if no file prefix in FULNAM
NAME	C*(*)	O	file name NAME = ' ', if no file name in FULNAM
EXTEN	C*(*)	O	file extension EXTEN = ' ', if no file extension in FULNAM

Here are examples for file names you might encounter on PC, Unix and Vax systems respectively:

```
          FULNAM                        PREFIX            NAME     EXTEN

    'C:\SEMPER\JUNK.DSK'          = 'C:\SEMPER\'       + 'JUNK' + '.DSK'
    '/usr/semper6/junk.dsk'       = '/usr/semper6/'    + 'junk' + '.dsk'
    '[user.sem.data]junk.dsk;4'   = '[user.sem.data]'  + 'junk' + '.dsk'
```

If any component is not present in FULNAM, a blank string ' ' is returned for that component.

FILDCD returns .TRUE. in case of error.

See also FILMAK.

LOGICAL FUNCTION FILDIR(DIR)

Return current directory as a string in DIR.

Argument	Type	I/O	
DIR	C*(*)	O	current directory path name

DIR must be long enough to receive the current directory path name up to a maximum of 255 characters. If the path name for the current directory is longer than 255 characters, FILDIR will fail.

FILDIR returns .TRUE. in case of error (e.g. DIR of insufficient length).

```
LOGICAL FUNCTION FILEXI(FULNAM,EXISTS)

Checks if the named file exists and is readable.

Argument    Type      I/O

FULNAM      C*(*)     I      fully specified file name

EXISTS      L           O    EXISTS = .TRUE.,  file exists and is readable
                                   = .FALSE., otherwise
```

Here are examples of fully specified names you might encounter on PC, Unix and Vax systems respectively:

```
'A:\SEMPER\GALLERY\PICTURE.DSK'
'/usr/demo/amoeba.dsk'
'[semper6.user]junk.dsk'
```

FILEXI returns .FALSE. unless an error occurs.

See also FILMAK and FILSEA.

```
LOGICAL FUNCTION FILMAK(NAME,DEFALT,FULNAM)
```

Builds a fully specified file name from a file name and a default specification.

Argument	Type	I/O	
NAME	C*(*)	I	file name, with or without directory prefix and file extension
DEFALT	C*(*)	I	default file name
PATH	C*(*)	O	fully specified file name

Any unspecified components (directory prefix, file name, file extension) of NAME are replaced by the corresponding components of DEFALT. If DEFALT does not contain a directory prefix then the current directory is used. The resulting fully specified file name (directory + name + extension) is then returned in FULNAM.

FILMAK imposes the following limits on the lengths of the components of a file name:

directory	130
name	80
extension	40

which means that the length of the string returned in FULNAM cannot exceed 250 characters.

FILMAK returns .TRUE. if an error is detected (for example, FULNAM too short to accommodate full file specification, badly specified file name (NAME=' ' and DEFALT=' ')).

See also FILDCD, FILDIR, FILSEA and FILOPN.

```
LOGICAL FUNCTION FILOPN(NAME,DEFALT,TYPE,INTENT,VERS,RECL,PATH,HANDLE)
```

Opens a named file and returns a file handle. Unspecified portions of the file name are taken in turn from the default filename DEFALT, the current directory and finally the Semper search path (see FILPAT).

Argument	Type	I/O	
NAME	C*(*)	I	name of file to be opened, (e.g. "FRED")
DEFALT	C*(*)	I	default file specification (e.g. "A:.DSK")
TYPE	I	I	TYPE = 1, character file = 2, binary file (currently not supported)
INTENT	I	I	INTENT = 1 for read access = 2 for write access = 3 for read/write access
VERS	I	I	if INTENT = 2 or 3: VERS = 1, replace old version of file, if it exists = 2, open file if it doesn't exist, otherwise fault existence of file = 3, re-use old version of file if it exists
RECL	I	I	record length (currently ignored)
PATH	C*(*)	O	full file name, if file successfully opened
HANDLE	I	O	file handle

FILOPN returns .TRUE. in case of error (e.g. no file found and INTENT=2).

When opening a file for reading (INTENT=1 or 3), the file is located using FILSEA.

See also FILCIO, FILCLS, FILDIR, FILPAT, FILMAK and FILSEA.

LOGICAL FUNCTION FILPAT(N,PATH)

Returns the specified component of the current system search path.

Argument	Type	I/O	
N	I	I	index into current search path
PATH	C*(*)	O	component of search path
			PATH = ' ', N is out of range

The current system search path defines a series of directories in which to look for a file. FILPAT returns in PATH the Nth directory in the search path. If N is out of range, a blank string is returned. The system search path is usually obtained from the environment variable called PATH.

Here is an example of FILPAT's operation on PC systems. If the current search path is "C:;C:\SEMPER", FILPAT will return the following:

```
N < 1, PATH = ' '
N = 1, PATH = 'C:'
N = 2, PATH = 'C:\SEMPER'
N > 2, PATH = ' '
```

For PC systems running MS-DOS, FILPAT imposes a limit of 255 characters on the length of the search path string (this is not a problem because MS-DOS's search path is limited to 127 characters anyway). On other systems FILPAT can process search path strings with lengths of up to 1024 characters. If the search path exceeds the limit, the string will be truncated and may cause FILPAT to return an error.

FILPAT returns .TRUE. in case of error.

Semper's command SHOW PATH lists the file search path that Semper uses to locate files. This always includes the current directory as the first directory in the search path.

See also FILSEA.

LOGICAL FUNCTION FILSEA(NAME,DEFALT,FULNAM,FOUND)

Searches for a file with name NAME and default file name DEFALT. If a file is found which is readable, FILSEA sets FOUND to .TRUE. and returns the fully specified file name for the file in FULNAM.

Argument	Type	I/O	
NAME	C*(*)	I	file name, with or without directory prefix and file extension
DEFALT	C*(*)	I	default file name
FULNAM	C*(*)	O	fully specified file name If FOUND = .FALSE., FULNAM = ' '
FOUND	L	O	FOUND = .TRUE., file exists and is readable = .FALSE., otherwise

Any unspecified components (directory prefix, file name, file extension) of NAME are replaced by the corresponding components of DEFALT.

If the resulting file name includes a directory prefix, FILSEA looks to see whether the file exists and is readable (see FILEXI). If the file name does not include a directory prefix, FILSEA searches the current directory (see FILDIR) and then directories in the system's search path (see FILPAT). Any existing file which is not readable is ignored. If a readable file is found, FOUND is set to .TRUE. and the fully specified file name is returned in FULNAM.

For example, on a PC system running MS-DOS, FILSEA would locate the file D:\SEMPER\PICTURES.DSK if NAME = 'PICTURES', DEFALT = 'IMAGES.DSK', D:\SEMPER is either the current directory or one of the directories in MS-DOS's search path and the file is readable.

FILSEA imposes the following limits on the lengths of the components of a file name:

directory	130
name	80
extension	40

which means that the length of the string returned in FULNAM cannot exceed 250 characters. For the limit imposed on the length of the system's search path, see FILPAT.

FILSEA returns .FALSE. unless an error is detected (e.g. file specification will not fit in FULNAM)

See also FILDCD, FILDIR, FILPAT and FILEXI.

```
LOGICAL FUNCTION FILSTR(PROMPT,FILE,N,LWRITE)
```

Returns in FILE the file name associated with the Semper NAME key. If the AGAIN option is set and LWRITE is .FALSE. (input file), FILSTR will return the last input file name obtained by a previous call to FILSTR. In the absence of both the NAME key and the AGAIN option, FILSTR will prompt the user for the file name. PROMPT specifies the prompt string.

Argument	Type	I/O	
PROMPT	C*(*)	I	prompt string PROMPT = ' ', the following prompt string is used: if LWRITE = .FALSE., 'Input file name' = .TRUE., 'Output file name'
FILE	C*(*)	O	file name string
N	I	O	length of file name string (maximum = 255) N = 0, no file name > 0, number of characters returned in FILE
LWRITE	L	I	LWRITE = .FALSE., fetch input file name = .TRUE., fetch output file name

Call FILSTR to obtain a file name. File names are specified in a command with the NAME text key. When the NAME key is used to specify an input file name (LWRITE = .FALSE.), the file name string is stored away by FILSTR for later use. The same input file can be specified in subsequent commands by substituting the option AGAIN for the NAME key.

If the NAME key or AGAIN option is not present or if the AGAIN option is specified and no input file name was previously specified, FILSTR will prompt for the file name. You can provide a prompt string in PROMPT. If PROMPT is blank, a default prompt of "Input file name" for an input file and "Output file name" for an output file is provided. FILSTR also appends the string " (as a textstring): " to the prompt string. If no file name is given in response to the prompt, FILSTR returns no file name (N = 0).

The NAME key must be followed by a properly formatted textstring (a series of quoted strings and/or arbitrary numerical expressions separated by commas). If the textstring is incorrectly formatted, FILSTR returns .TRUE. and reports the fault with Semper error 3 - "Bad value for NAME".

```
LOGICAL FUNCTION FSAMPL(XMIN,XMAX,YMIN,YMAX,X,DX,NCOL,Y,DY,NROW)
```

Returns parameters defining a regular sampling grid which defines the positions in graphics coordinates of all the display pixels that lie within the subregion given by XMIN, XMAX, YMIN, and YMAX. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XMIN	R	I	subregion limits in graphics coordinates
XMAX	R	I	.
YMIN	R	I	.
YMAX	R	I	.
X	R	O	graphics coordinates of top left pixel lying within
Y	R	O	the subregion
DX	R	O	increment in graphics coordinates between adjacent
DY	R	O	pixels in horizontal and vertical directions
NCOL	I	O	number of pixels within horizontal limits
NROW	I	O	number of pixels within vertical limits

FSAMPL returns parameters defining a grid which coincides with all the display pixel positions that fall within the specified subregion. All the display pixels in the region can then be accessed at their precise locations via the graphics coordinates (XPIXEL,YPIXEL) in the following manner:

```
      IF (FSAMPL(XMIN,XMAX,YMIN,YMAX,X,DX,NCOL,Y,DY,NROW)) RETURN

      DO 20 J = 1,NROW
         YPIXEL = Y + REAL(J-1) * DY
         DO 10 I = 1,NCOL
            XPIXEL = X + REAL(I-1) * DX
            <access pixel with graphics coordinates (XPIXEL,YPIXEL)>
   10    CONTINUE
   20 CONTINUE
```

The above loop is guaranteed to access the region from left to right and top to bottom.

If the X or Y dimension of the subregion exceeds 30000, FSAMPL returns .TRUE. and reports this with error 89.

FSAMPL returns .FALSE. unless an error occurs (e.g. region too large).

See also FSREGN.

```
LOGICAL FUNCTION FSARC(XCEN,YCEN,RADIUS,THETA1,THETA2)
```

Draws an arc in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XCEN	R	I	centre of the arc in graphics coordinates
YCEN	R	I	.
RADIUS	R	I	arc radius in graphics units
THETA1	R	I	angles between the positive X axis and the lines
THETA2	R	I	joining the start and end points of the arc to the centre of the arc, measured anticlockwise in radians

The arc is drawn in an anti-clockwise direction from angular position THETA1 to angular position THETA2.

The arc is clipped at the graphics limits (see FSQLIM).

FSARC returns .FALSE. if successful.

```
LOGICAL FUNCTION FSARRO(XTAIL,YTAIL,XHEAD,YHEAD)
```

Draws an arrow from (XTAIL,YTAIL) to (XHEAD,YHEAD) in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XTAIL	R	I	position of arrow tail in graphics coordinates
YTAIL	R	I	.
XHEAD	R	I	position of arrow head in graphics coordinates
YHEAD	R	I	.

The arrow is clipped at the graphics limits (see FSQLIM).

FSARRO returns .FALSE. when successful.

```
LOGICAL FUNCTION FSBAND(IOP,X,Y,N,CLOSED)
```

FSBAND provides a basic rubber-banding facility. It draws or undraws an open curve (poly-line) or closed curve (polygon) in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
IOP	I	I	IOP = 1, draw curve = 2, undraw curve
X	R(*)	I	curve vertices in graphics coordinates
Y	R(*)	I	.
N	I	I	number of vertices specifying the curve
CLOSED	L	I	CLOSED = .TRUE., closed curve (end point is joined to start point) = .FALSE., open curve

FSBAND draws curves in much the same way as FSCURV, but it is also able to undraw them. Rubberbanding is effected by making a rapid series of calls to FSBAND to draw and undraw graphical objects. A short pause after drawing an object and prior to undrawing the same object should be introduced by calling WAITS otherwise the object will tend to flicker.

Drawing and undrawing may be achieved on some systems by XORing the curve with the data in the overlay plane. If this is the case, any overlap between different parts of the curve will cancel out. Also, any graphical output occuring after FSBAND has drawn an object, and before it has undrawn the same object, will intefere with the undrawing process. With care it should be possible to avoid the worst side-effects of XORing.

Graphical output is clipped at the monitor limits (see FSQMON).

FSBAND draws and undraws on the real part of a complex display picture unless the option IM is set when FSINIT is called.

FSBAND returns .FALSE. if successful.

See also FSDRAG.

```
LOGICAL FUNCTION FSBORD( )
```

Marks the border of the picture, partition or frame (depending on the current graphics coordinate system) in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

FSBORD returns .FALSE. if successful.

```
LOGICAL FUNCTION FSCIRC(XCEN,YCEN,RADIUS)
```

Draws a circle in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XCEN	R	I	centre of the circle in graphics coordinates
YCEN	R	I	.
RADIUS	R	I	circle radius in graphics units

The circle is clipped at the graphics limits (see FSQLIM).

FSCIRC returns .FALSE. if successful.

A call to FSCIRC is functionally equivalent to:

```
      IF (FSARC(XCEN,YCEN,RADIUS,0.0,TWOPI)) RETURN
```

```
LOGICAL FUNCTION FSCTYP(ITYPE)
```

Selects the appearance of the display cursor (the cursor type). If FSCTYP is called while a cursor is switched on, the appearance of the cursor will change immediately on the display.

Argument	Type	I/O	
ITYPE	I	I	cursor type

```
ITYPE = 0, invisible/hidden cursor
      = 1, small cross
      = 2, medium cross
      = 3, large cross
      = 4, full screen cross-hair
```

The cursor types 0 to 4 are supported on all platforms. Some implementations of the underlying display software may support additional cursor types.

FSCTYP returns .TRUE. in case of error.

See also FSCURS.

```
LOGICAL FUNCTION FSCURS(IOP,X,Y)
```

Controls the display and positioning of the cursor. The cursor must first be switched on and finally switched off with separate calls to FSCURS with IOP set to 1 and 3 respectively. In between these calls, the cursor is repositioned by calling FSCURS with IOP set to 2. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
IOP	I	I	IOP = 1, switch on the cursor and display it at position (X,Y) = 2, move the cursor to position (X,Y) = 3, switch off the cursor at position (X,Y)
X	R	I	new cursor position in graphics coordinates (will be
Y	R	I	ignored by some systems if IOP = 3)
		O	actual cursor position, clipped at the current
		O	monitor limits

The appearance of the cursor is determined by the cursor type (set by calling FSCTYP). In particular the cursor may be made to disappear while it is switched on by selecting cursor type 0.

On some systems the cursor is displayed and erased by XORing it with the data in the overlay plane. To avoid any unpleasant side-effects, the cursor should be hidden, by calling FSCTYP(0), when any annotation is being output to the overlay plane, and re-instated afterwards.

The cursor is positioned at (X,Y) in graphics coordinates, unless this position falls outside the current monitor limits (see FSQMON), in which case the position is clipped at the monitor limits. X and Y always return the actual (clipped) cursor position.

When switching off the cursor (IOP = 3), the supplied position may be used to position the host system's cursor if appropriate. It is not an error to switch the cursor on or off more times than is required.

FSCURS returns .TRUE. in case of error (e.g. attempt to move the cursor (IOP = 2) without first switching it on (IOP = 1)).

```
LOGICAL FUNCTION FSCURV(X,Y,N,CLOSED)
```

Draw an open curve (poly-line) or closed curve (polygon) in the overlay plane. The curve is drawn as a series of straight lines joining the vertices defined in arrays X and Y. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
X	R(*)	I	curve vertices in graphics coordinates
Y	R(*)	I	.
N	I	I	number of vertices defining curve
CLOSED	L	I	CLOSED = .TRUE., closed curve (end point is joined to start point)
			= .FALSE., open curve

The curve is clipped at the graphics limits (see FSQLIM).

FSCURV returns .FALSE. if successful.

```
LOGICAL FUNCTION FSDRAG(IOP,X0,Y0,DX,DY,N,CLOSED)
```

FSDRAG provides a basic animation facility. It draws or undraws an open curve (poly-line) or closed curve (polygon) in the overlay plane. The curve is specified as a series of offsets relative to the anchor position (X0,Y0). FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
IOP	I	I	IOP = 1, draw curve = 2, undraw curve
X0	R	I	position of anchor point in graphics coordinates
Y0	R	I	.
DX	R(*)	I	offsets in graphics units relative to the anchor
DY	R(*)	I	point defining curve vertices
N	I	I	number of vertices specifying the curve
CLOSED	L	I	CLOSED = .TRUE., closed curve (end point is joined to start point) = .FALSE., open curve

FSDRAG draws curves in much the same way as FSCURV, but it is also able to undraw them. Since the vertiqes are specified as offsets relative to the anchor point (X0,Y0), movement of fixed graphical objects can easily be achieved simply by altering X0 and Y0.

Animation is effected by making a rapid series of calls to FSDRAG to draw and undraw an object. There should be a short pause between drawing and undrawing an object (with a call to WAITS), otherwise the object will tend to flicker.

Drawing and undrawing may be achieved on some systems by XORing the curve with the data in the overlay plane. If this is the case, any overlap between different parts of the curve will cancel out. Also, any graphical output occuring after FSDRAG has drawn an object, and before it has undrawn the same object, will intefere with the undrawing process. With care it should be possible to avoid the worst side-effects of XORing.

Graphical output is clipped at the monitor limits (see FSQMON).

FSDRAG draws and undraws on the real part of a complex display picture unless the option IM is set when FSINIT is called.

FSDRAG returns .FALSE. if successful.

A call to FSDRAG is functionally equivalent to:

```
      DO 10 I=1,N
         XX(I)=DX(I)+X0
         YY(I)=DY(I)+Y0
   10 CONTINUE

      IF (FSBAND(IOP,XX,YY,N,CLOSED)) RETURN
```

```
LOGICAL FUNCTION FSELEC( )
```

Records the nature of the current graphics coordinate system by setting the Semper variable DISPLAY or variables FS and CFRAME.

If the current graphics coordinates are picture based or partition based, DISPLAY is set to the corresponding display picture or partition number. Otherwise FS is set to the current display device and CFRAME is set to the frame number.

FSELEC returns .FALSE. if successful.

```
LOGICAL FUNCTION FSERAS(IOP,XMIN,XMAX,YMIN,YMAX)
```

Erases a region of the display. The value of IOP determines whether the region is erased in the image plane, in the overlay plane, or both. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
IOP	I	I	IOP = 1, erase image plane only = 2, erase overlay plane only = 3, erase image and overlay planes
XMIN	R	I	limits of the region to be erased, in graphics coordinates
XMAX	R	I	.
YMIN	R	I	.
YMAX	R	I	.

The region erased is confined to the graphics limits (see FSQLIM).

FSERAS returns .TRUE. if an error is detected.

```
LOGICAL FUNCTION FSFLUS( )
```

Flushes the graphics buffers, transmitting any buffered graphical output to the graphics hardware.

FSFLUS is automatically called by the command interpreter at the end of a Semper command if there is any unflushed graphical output.

FSFLUS returns .FALSE. unless an error occurs.

```
LOGICAL FUNCTION FSINIT(MODE,NDIS)
```

FSINIT initialises the graphics coordinate system for subsequent graphical operations via the FS.... routines.

Argument	Type	I/O	
MODE	I	I	desired graphics coordinate system to be initialised MODE = 1, frame based coordinates = 2, partition based coordinates = 3, picture based coordinates
NDIS	I	I	if MODE = 1, display frame number = 2, display partition number, with optional device component = 3, display picture number, with optional device component

If a frame based coordinate system is selected (MODE = 1), the origin lies at the centre of the frame, with X increasing from left to right, Y increasing upwards, and with unit scaling in both directions.

If a partition based coordinate system is selected (MODE = 2), the origin lies at the centre of the partition, with X increasing from left to right, Y increasing upwards, and with unit scaling in both directions.

If a picture based coordinate system is selected (MODE = 3), the alignment and scaling of the coordinate axes depends on the type of display picture that is being addressed. There are four types of display pictures: images, graphs, histograms and YMOD plots. Any attempt to access a display picture which is a YMOD plot will be faulted by FSINIT with error 48.

For displayed images, the origin is aligned with the centre column and row of the picture. X increases from left to right and Y increases upwards. The axes are scaled so that unit steps in X and Y correspond exactly to unit steps in the original source image. If the image was displayed with undersampling or magnified, graphical output will be scaled so that it appears in the same place with respect to the display picture.

Complex images are displayed as a pair of images, one for the real part and one for the imaginary part. The two images are displayed side by side with the real part on the left and the imaginary part on the right. By default the origin is aligned with the real part and graphical output will appear in the real part and be duplicated in the imaginary part (with the exception of rubber-band and cursor output which is not duplicated - see FSBAND, FSCURS and FSDRAG). If the general option RE is set, the origin remains aligned with the real part but graphical output will appear only in the real part. If the general option IM is set, the origin is aligned with the imaginary part and graphical output will appear only in the imaginay part.

For graphs, the picture axes are arranged so that the X origin and scaling is the same as for an image. In the Y direction, the origin and scaling are arranged so that Y ordinate values agree with the scale of values for the graph. Note that the Y axis may be inverted (Y increasing downwards) if the graph was displayed with the command DISPLAY NEGATED. The vertical scaling can be extreme if the graph encompases a large range of values.

For histograms, the picture axes are arranged so that X ordinate values correspond to histogram bin values (the original picture values associated with each bin) and Y ordinate values correspond to bin counts.

All graphical output is clipped. The clipping limits are as follows:

frame coordinates	-	frame limits
partition coordinates	-	partition limits
picture coordinats	-	limits of the associated partition

The clipping limits can be obtained by calling FSQLIM. Output for FSBAND and FSDRAG is clipped at the monitor limits (the limits of the region visible on the display). The monitor limits can be obtained by calling FSQMON.

The border limits define the limits of the frame, partition or picture, as appropriate. These can be obtained by calling FSQBOR and displayed by calling FSBORD. Note that the clipping limits are not the same as the border limits for picture based graphics. This allows annotation to appear around a display picture as long as it does not run off the edge of the associated display partition.

WARNING: When SEMOPN is called to open a display picture, it makes a call to FSINIT, which will intefere with any previous call to FSINIT. In this case a second call to FSINIT should be made to re-establish the desired graphics coordinate system.

FSINIT returns .TRUE. in case of error (frame number out of range, no display picture, device not a display device, display picture is a YMOD plot, etc.).

```
LOGICAL FUNCTION FSLINE(X1,Y1,X2,Y2)
```

Draws a line from (X1,Y1) to (X2,Y2) in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
X1	R	I	start point of the line in graphics coordinates
Y1	R	I	.
X2	R	I	end point of the line in graphics coordinates
Y2	R	I	.

Lines are truncated at the graphics limits (see FSQLIM).

FSLINE returns .FALSE. when successful.

```
LOGICAL FUNCTION FSLIST(X,Y,N,MODE,SIZE)
```

Marks a series of positions in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
X	R(*)	I	array of positions to be marked, in graphics
Y	R(*)	I	coordinates
N	I	I	number of positions to be marked
MODE	I	I	style of symbol to be used
			MODE = 1, horizontal/vertical cross
			= 2, diagonal cross
			= 3, square
			= 4, diamond
			= 5, single pixel (SIZE is ignored)
SIZE	I	I	symbol radius in pixel units

Points outside the graphics limits (see FSQLIM) are not marked.

FSLIST returns .FALSE. if successful.

A call to FSLIST is functionally equivalent to:

```
      DO 10 I=1,N
         IF (FSMARK(X(I),Y(I),MODE,SIZE)) RETURN
   10 CONTINUE
```

```
LOGICAL FUNCTION FSLURW(IOP,LUTN,MODE,LUTBUF)
```

Transfers the look-up table data in LUTBUF from/to look-up table LUTN in the display hardware.

Argument	Type	I/O	
IOP	I	I	IOP = 1, read data from hardware look-up table LUTN = 2, write data to hardware look-up table LUTN
LUTN	I	I	hardware look-up table number
MODE	I	I/O	look-up table type MODE = 1, monochrome look-up table = 2, false colour look-up table = 3, full colour look-up table
LUTBUF I(LUTLEN,3)		I/O	array containing look-up table data

When writing look-up table data to the display hardware, make sure that the data is restricted to the range 0 to LUTMAX.

Semper stores look-up table data in the display as well as on the work disc. Any changes brought about by a call to FSLURW must be reflected on the work disc by making a corresponding call to SEMLUT.

Monochrome look-up table data are accessed in the array LUTBUF in the following manner:

```
      DO 10 I=1,LUTLEN
         GREY(I)=LUTBUF(I,1)
   10 CONTINUE
```

False and full-colour look-up table data are accessed in the array LUTBUF as follows:

```
      DO 10 I=1,LUTLEN
         RED(I)  =LUTBUF(I,1)
         GREEN(I)=LUTBUF(I,2)
         BLUE(I) =LUTBUF(I,3)
   10 CONTINUE
```

FSLURW returns .FALSE. unless an error is detected.

```
LOGICAL FUNCTION FSMARK(X,Y,MODE,SIZE)
```

Marks a position in the overlay plane. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
X	R	I	position to be marked in graphics coordiantes
Y	R	I	.
MODE	I	I	style of symbol to be used
			MODE = 1, horizontal/vertical cross
			= 2, diagonal cross
			= 3, square
			= 4, diamond
			= 5, single pixel (SIZE is ignored)
SIZE	I	I	symbol radius in pixel units

A position outside the graphics limits (see FSQLIM) is not marked.

FSMARK returns .FALSE. if successful.

See also FSLIST.

```
LOGICAL FUNCTION FSOPTN(MODE,NDIS)
```

FSOPTN is used in conjunction with the syntax "$1= frame partition picture". It sets MODE according to whether the option FRAME, PARTITION or PICTURE is set. If none of these options is set, option PICTURE is assumed. NDIS is set to the value of the $1 key.

Argument	Type	I/O	
MODE	I	O	MODE = 1, option FRAME is set = 2, option PARTITION is set = 3, option PICTURE is set or no option set
NDIS	I	O	value of $1 key if set, otherwise the value of the variable CFRAME (MODE = 1) or DISPLAY (MODE = 2 or 3)

The values returned by FSOPTN are suitable for passing to FSINIT, that is

```
IF(FSOPTN(MODE,NDIS)) RETURN
IF(FSINIT(MODE,NDIS)) RETURN
```

establishes the graphics coordinate system specified on the command line in accordance with the FRAME, PARTITION and PICTURE options, for example,

```
frame 1                 establishes frame based coordinates
partition 4             establishes partition based coordinates
picture display         establishes picture based coordinates
```

FSOPTN returns .TRUE. if an error is detected (e.g. if more than one of the options FRAME, PARTITION, PICTURE is set).

```
LOGICAL FUNCTION FSOVRW(IOP,IROW,N,X,Y)
```

Reads data from or writes data to the overlay plane. The data transferred consist of N binary pixel values stored in IROW. Zero values of IROW correspond to overlay pixels that are turned off, non-zero values to pixels that are turned on. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
IOP	I	I	IOP = 1, read data from the overlay plane = 2, write data to the overlay plane
IROW	I(*)	O	if IOP = 1, row of data with zero values returned for pixels where the overlay is off or for pixels falling outside the graphics limits and non-zero values returned otherwise
		I	= 2, overlay pixels turned on for non-zero data values lying within the graphics limits, otherwise pixels turned off
N	I	I	number of data values in IROW
X	R	I	start position of row in graphics coordinates
Y	R	I	

Writing to, or reading from, a region that exceeds the graphics limits is allowed. When reading (IOP = 1), FSOVRW zero pads, and when writing (IOP = 2), FSOVRW ignores, any part of IROW that falls outside the graphics limits (see FSQLIM).

The effect of writing to or reading from a complex display picture depends on the setting of the options RE and IM when FSINIT is called. If neither option is set, FSOVRW reads from the real part and writes to both parts. If option RE is set FSOVRW accesses just the real part, and if the option IM is set, only the imaginary part is accessed.

N data values are transfered starting at graphics coordinates (X,Y), stepping from left to right, with adjacent display pixels in the overlay plane corresponding to consecutive elements of IROW.

FSOVRW returns .FALSE. unless error is detected.

```
LOGICAL FUNCTION FSQBOR(XMIN,XMAX,YMIN,YMAX)
```

Returns border limits in graphics coordinates. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XMIN	R	O	border limits in graphics coordinates such that
XMAX	R	O	XMIN < XMAX and YMIN < YMAX
YMIN	R	O	.
YMAX	R	O	.

The border limits correspond to the frame limits in frame based coordinates, to the partition limits for partition based coordinates, and to the picture limits for picture based coordinates.

FSQBOR returns .FALSE. if successful.

See also FSQLIM.

```
LOGICAL FUNCTION FSQCHA(CHORIZ,CHVERT)
```

Returns the character width and height, i.e. the character cell size. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
CHORIZ	R	O	width of character cell in graphics units
CHVERT	R	O	height of character cell in graphics units

FSQCHA returns .FALSE. unless an error occurs.

```
LOGICAL FUNCTION FSQLIM(XMIN,XMAX,YMIN,YMAX)
```

Returns the graphics clipping limits, i.e. the limits beyond which any graphical output is clipped. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XMIN	R	O	clipping limits in graphics coordinates such that
XMAX	R	O	XMIN < XMAX and YMIN < YMAX
YMIN	R	O	.
YMAX	R	O	.

The clipping limits correspond to the frame limits for frame based graphics coordinates, to the partition limits for partition based coordinates, and to the limits of the associated partition for picture based coordinates.

FSQLIM returns .FALSE. when successful.

See also FSQBOR.

```
LOGICAL FUNCTION FSQMON(FRAME,ZOOM,XMIN,XMAX,YMIN,YMAX)
```

Returns the current viewing conditions defining which display frame is visible, the current zoom factor and which region of the display is visible (monitor limits). FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
FRAME	I	O	currently visible display frame
ZOOM	I	O	current zoom factor
XMIN	R	O	current monitor limits in graphics coordinates such
XMAX	R	O	that XMIN < XMAX and YMIN < YMAX
YMIN	R	O	.
XMAX	R	O	.

The current viewing conditions may differ from those requested (see FSVIEW) according to any limitations of the display hardware (e.g. truncation of pan offsets, limited number of zoom factors, etc.).

FSQMON returns .FALSE. if successful.

```
LOGICAL FUNCTION FSQTRA(XSCALE,XOFF,YSCALE,YOFF)
```

Returns the parameters that define the transformation between graphics coordinates and display coordinates. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XSCALE	R	O	X scale factor
XOFF	R	O	X offset
YSCALE	R	O	Y scale factor
YOFF	R	O	Y offset

A position (XG,YG) in graphics coordinates is converted to the corresponding position (XD,YD) in display coordinates in the following way:

```
XD = NINT( XSCALE * XG + XOFF )
YD = NINT( YSCALE * YG + YOFF )
```

Note that display coordinates are limited to discrete integer values which address individual display pixels. The origin (0,0) for display coordinates is at the top left of a display frame.

FSQTRA returns .FALSE. if successful.

LOGICAL FUNCTION FSREGN(XMIN,XMAX,YMIN,YMAX)

Returns the limits of the sub-region defined with respect to the current graphics area by the 2D sub-region keys SIZE, SI2, POSITION and PO2, and the options LEFT/RIGHT and TOP/BOTTOM. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
XMIN	R	O	sub-region limits in graphics coordinates such that
XMAX	R	O	XMIN < XMAX and YMIN < YMAX
YMIN	R	O	.
YMAX	R	O	.

The sub-region limits are clipped at the graphics border limits (see FSQBOR).

FSREGN returns .TRUE. in case of error (sub-region completely outside border limits, zero or negative sub-region size, conflicting options).

See also GETRG1.

LOGICAL FUNCTION FSROW(IOP,IROW,N,X,Y)

Reads data from or writes data to the image plane of the display. The data transfered consist of N integer values stored in IROW. The data values are not scaled. FSINIT must be called first to establish the graphics coordinate system.

Argument	Type	I/O	
IOP	I	I	IOP = 1, read data from the display = 2, write data to the display
IROW	I(*)	O	if IOP = 1, row of actual pixel values or zero for any pixels falling outside the graphics limits
		I	= 2, row of actual pixel values to be written directly to the display, ignoring any pixels that lie outside the graphics limits
N	I	I	number of data values in IROW to be transfered
X	R	I	start position of row in graphics coordinates
Y	R	I	.

When reading (IOP = 1), FSROW zero pads and when writing (IOP = 2), FSROW clips any part of IROW that falls outside graphics limits (see FSQLIM).

The data values are not scaled and only values in the range 0 to LUTLEN-1 can be stored in the display. On output, any values outside this range will be truncated.

The effect of writing to or reading from a complex display picture depends on the setting of the options RE and IM when FSINIT is called. If neither option is set, FSROW reads from the real part and writes to both parts. If option RE is set FSROW accesses just the real part, and if the option IM is set, only the imaginary part is accessed.

N data values are transfered starting at graphics position (X,Y) and stepping from left to right with adjacent display pixels corresponding to consecutive elements of IROW.

FSROW returns .FALSE. unless an error is detected.

See also FSOVRW and SEMROW.

```
LOGICAL FUNCTION FSTEXT(ITEXT,N,X,Y,XJUST,YJUST)
```

Draws a line of text to the overlay plane. FSINIT must be called first to establish a graphics coordinate system.

Argument	Type	I/O	
ITEXT	I(*)	I	array containing integer ASCII codes representing the text to be output
N	I	I	number of characters in ITEXT
X	R	I	position in graphics coordinates at which the text
Y	R	I	is to be displayed
XJUST	I	I	text justification in the X direction
			XJUST < 0, left justification = 0, centre justification > 0, right justification
YJUST	I	I	text justification in the Y direction
			YJUST < 0, bottom justification = 0, centre justification > 0, top justification

The string of N characters will appear in a position relative to the graphics coordinates (X,Y) determined by XJUST and YJUST. The characters are clipped at the graphics limits (see FSQLIM).

Call SEMICS to convert a FORTRAN character string to an array of integer ASCII codes.

A call to FSQCHA will return the size of the character cell in graphics units.

FSTEXT returns .FALSE. unless an error is detected.

```
LOGICAL FUNCTION FSVIEW( )
```

Sets the viewing conditions to make visible the current graphics area. FSINIT must be called first to establish the graphics coordinate system.

FSVIEW reads the keys LUT, ZOOM and PAN, PA2 and attempts to set up the viewing conditions accordingly. These viewing conditions may not always be met due to limitations in the display hardware. A call to FSQMON will return the viewing conditions currently maintained by the display hardware.

FSINIT provides a default look-up table number and a default view centre. The default for the zoom factor is always 1. FSVIEW sets up the viewing conditions to the defaults established by FSINIT, modified according to the keys LUT, ZOOM, PAN, PA2 if these are set.

The main use of FSVIEW is to switch the viewing conditions when the option VIEW is set as follows:

```
      LOGICAL LVIEW

      LVIEW=OPT(IPACK('VIEW'))
      IF(LVIEW) THEN
         IF(FSVIEW()) RETURN
      ENDIF
```

FSVIEW returns .TRUE. if an error is detected (e.g. bad look-up table number, non-existent look-up table, zero or negative zoom factor).

```
LOGICAL FUNCTION FSXWIR(X0,Y0,X,Y)
```

Switches on the display cursor at the position (X0,Y0) from where it can then be repositioned under the control of the pointing device. The final cursor position is returned in X and Y. FSINIT must be called first to establish a graphics coordinate system.

Argument	Type	I/O	
X0	R	I	initial cursor position in graphics coordinates
Y0	R	I	.
X	R	O	final cursor position in graphics coordinates
Y	R	O	.

If the initial cursor position falls outside the current monitor limits (see FSQMON), the cursor will appear on the display at the center of the monitor limits.

For a complex display picture, the operation of FSXWIR depends on the setting of the options RE and IM when FSINIT is called. If neither option is set, or if the option RE is set, the cursor is displayed with respect to the real part of the display picture. If option IM is set, the cursor is displayed with respect to the imaginary part.

FSXWIR returns .FALSE. unless an error is detected.

```
SUBROUTINE FT1D(F,N,SIGN,HPLANE,MODSQ,LN)
```

Effects in-place a 1D Fourier transform on the array real F, optionally calculating the power spectrum or log(power spectrum). The transform is not normalised.

Argument	Type	I/O	
F	R(*)	I	array of data to be transformed
	CP(*)	O	transformed result
N	I	I	number of points, must be a power of two
SIGN	I	I	SIGN = -1, transform from real to Fourier space = +1, perform inverse transform (unnormalised)
HPLANE	L	I	HPLANE = .TRUE., real, half-plane transform (real to complex) = .FALSE., complex, full-plane transform (complex to complex)
MODSQ	L	I	MODSQ = .TRUE., perform modulus-squared calculation SIGN = -1, modulus-squared of the data is calculated after the forward transform = +1, modulus-squared of the data is calculated before the inverse transform
LN	L	I	LN = .TRUE. and SIGN = -1, the logarithm of the modulus-squared of the data is taken after the forward transform

If HPLANE is .TRUE., FT1D calculates the half plane transform of a real input image, with its origin at the centre. The transform is conjugate symmetric so only the right-hand half plane is retained, giving an complex output picture with origin at centre-left. If HPLANE is .FALSE., FT1D calculates a full plane Fourier transform of a complex image resulting in a complex output picture with its origin also at the centre.

FT1D can only transform data where the number of points is a power of 2.

See also FT2D, INVFT2.

```
LOGICAL FUNCTION FT2D(NPIC,CLASS,FORM,MSQ,LN)
```

Effects 2-D forward Fourier Transform from logical picture LP1 to picture number NPIC, returning logical picture number of the output picture in LP2. Options are provided to take the modulus-squared or log-modulus-squared of the transformed picture.

Argument	Type	I/O	
NPIC	I	I	fully specified output picture number
CLASS	I	I	output picture class

```
CLASS = NCLIMA, Image
      = NCLMAC, Macro
      = NCLFOU, Fourier
      = NCLSPE, Spectrum
      = NCLCOR, Correlation
      = NCLWAL, Walsh
      = NCLPLI, Position list
      = NCLLUT, Display look-up table
      = NCLHIS, Histogram
      = NCLUND, Undefined
```

FORM	I	I	output picture form

```
FORM =  NFMBYT, byte
     =  NFMINT, integer
     =  NFMFP,  floating point
     =  NFMCOM, complex
```

MSQ	L	I	MSQ = .TRUE., output the modulus-squared of the transform in place of the transform
LN	L	I	LN = .TRUE., if MSQ = .TRUE., output the logarithm of the modulus-squared

COMMON variables

Variable	Type	I/O	
LP1	I	I	logical picture number assigned to a suitable source picture (size must be factorisable)
LP2	I	O	logical picture number assigned to the output picture

Row buffers

Buffer	Type	I/O	
RBS			used as workspace

FT2D returns the Fourier transform of the source picture LP1 as picture LP2. If the source picture is complex, the output picture will have the same dimensions as the source picture. If the source picture is not complex, the Fourier transform has complex conjugate symmetry, so the output picture will contain only the right-hand half-plane of the transform.

The dimensions of the source picture must be factorisable (they must have factors 2, 3, 4 and 5 only and must be a multiples of 4). The Semper command SHOW SIZES lists all the factorisable sizes supported by FT2D.

The output picture form specified by FORM is imposed when the final result is output. If the final result is complex and the output form is not complex, the real part of the result is output and the imaginary part is discarded.

The output picture is assigned the class specified by CLASS - FT2D does not check the output picture class. It is the caller's responsibility to ensure that the class is appropriately specified. Some Semper commands will not work with Fourier transforms if the picture does not have an appropriate form, or they may produce different results according to the picture class (see commands FOURIER, IMAGE and PS).

WARNING: FT2D uses all of the row buffer array RBS. Data held in any of the row buffers will not be preserved.

FT2D returns .TRUE. if an error is detected (e.g. size of the source picture is not factorisable).

See also FT1D and INVFT2.

LOGICAL FUNCTION GENHST(LPN,NCHAN)

Constructs a histogram of pixel values for logical picture LPN, or a subregion thereof, using NCHAN bins covering the range specified by the Semper variables MIN and MAX. The result is written to the row buffer RB1.

Argument	Type	I/O	
LPN	I	I	logical picture number for the source picture
NCHAN	I	I	number of histogram channels or bins (NCHAN equal width bins span the range specified by the Semper variables MIN and MAX)

COMMON variables

Variable	Type	I/O	
SMGL1	L	I	SMGL1 = .FALSE., the whole picture is to be scanned = .TRUE., subregion to be scanned is defined by the limits in SMGI1 to SMGI6
SMGI1	I	I	first column of the picture included in the subregion
SMGI2	I	I	" row " " " " " " "
SMGI3	I	I	" layer " " " " " " "
SMGI4	I	I	last column of the picture included in the subregion
SMGI5	I	I	" row " " " " " " "
SMGI6	I	I	" layer " " " " " " "

Row buffers

Buffer	Type	I/O	
RB1	R(*)	O	buffer containing NCHAN histogram counts
RB2			used as workspace
RB3			used as workspace

The Semper variables MIN and MAX specify the source picture values which corresponds to the left hand edge of the first bin (bin number 1) and the right hand edge of the last bin (bin number NCHAN) respectively.

For complex pictures, the histograms counts are equal to the sum of the counts for the real and imaginary parts.

If SMGL1 is .FALSE., the whole of the source picture is scanned and the values of SMGI1 to SMGI6 are ignored. Otherwise, SMGI1 to SMGI6 define the limits of the subregion of the source picture to be scanned.

The COMMON variables SMGL1 and SMGI1 to SMGI6 can be set to appropriate values by calling the routine GETRG1 immediately before calling GENHST:

```
      IF (GETRG1(COL1,ROW1,LAY1,COL2,ROW2,LAY2,LPN)) RETURN

      SMGL1 = COL1 .NE. 1 .OR. COL2 .NE. NCOLS(LPN) .OR.
     +        ROW1 .NE. 1 .OR. ROW2 .NE. NROWS(LPN) .OR.
     +        LAY1 .NE. 1 .OR. LAY2 .NE. NLAYS(LPN)

      IF (SMGL1) THEN
         SMGI1 = COL1
         SMGI2 = ROW1
         SMGI3 = LAY1
         SMGI4 = COL2
         SMGI5 = ROW2
         SMGI6 = LAY2
      ENDIF

      IF (GENHST(LPN,NCHAN)) RETURN
```

GETRG1 reads the subregion keys SIZE, POSITION and LAYER and options LEFT/RIGHT, TOP/BOTTOM and NEAR/FAR.

WARNING: Row buffers RB2 and RB3 are used as workspace. Values held in these buffers will not be preserved.

GENHST returns .FALSE. unless an error is detected.

See also MEANSD, RANGE and SEMRNG.

```
LOGICAL FUNCTION GETFRM(IFORM,IDEF)
```

Returns in IFORM the form number corresponding to one of the options BYTE, INTEGER, FP or COMPLEX. If no options are set, IFORM is set equal to IDEF.

Argument	Type	I/O	
IFORM	I	O	IFORM = NFMBYT, if option BYTE is set = NFMINT, if option INTEGER is set = NFMFP , if option FP is set = NFMCOM, if option COMPLEX is set = IDEF, otherwise
IDEF	I	I	default value for IFORM IDEF = NFMBYT, byte IDEF = NFMINT, integer IDEF = NFMFP, floating-point IDEF = NFMCOM, complex

GETFRM returns .FALSE. unless conflicting options are detected (e.g. options FP and BYTE are set).

```
LOGICAL FUNCTION GETRG1(COL1,ROW1,LAY1,COL2,ROW2,LAY2,LPN)
```

Examines the 3D sub-region keys and options and returns the sub-region limits (start and end row, column and layer numbers) defined with respect to the picture with logical picture number LPN.

Argument	Type	I/O	
COL1	I	O	start and end column numbers for the sub-region
COL2	I	O	such that COL1 <= COL2
ROW1	I	O	start and end row numbers for the sub-region
ROW2	I	O	such that ROW1 <= ROW2
LAY1	I	O	start and end layer numbers for the sub-region
LAY2	I	O	such that LAY1 <= LAY2
LPN	I	I	logical picture number for the picture for which the sub-region is specified

The 3D sub-region keys (see section 6.5 of the Advanced Users' Guide) are:

```
SIZE, SI2
POSITION, PO2
LEFT/RIGHT
TOP/BOTTOM

SI3        }
PO3        }  or  LAYER, LA2
NEAR/FAR   }
```

The limits of the sub-region are truncated at the picture boundaries.

GETRG1 sets up the subregion information in COMMON for subsequent use by RANGE, MEANSD and GENHST.

GETRG1 returns .TRUE. in case of error (e.g. region completely outside picture).

See also GETRG2.

```
LOGICAL FUNCTION GETRG2(X,Y,LAY1,LAY2,UX,UY,VX,VY,NU,NV,LPN)
```

Examines the full set of sub-region keys and options (see the next page) and returns data defining a skewed and/or rotated sub-region with respect to the picture with logical picture number LPN.

Argument	Type	I/O	
X	R	O	picture coordinates of the first corner of the
Y	R	O	sub-region
LAY1	I	O	start and and layer numbers of the sub-region
LAY2	I	O	such that LAY1 <= LAY2
UX	R	O	vector in picture coordinates corresponding to a
UY	R	O	unit step in the U direction of the sub-region
VX	R	O	vector in picture coordinates corresponding to a
VY	R	O	unit step in the V direction of the sub-region
NU	I	O	number of points in the U direction
NV	I	O	number of points in the V direction
LPN	I	I	logical picture number for the picture for which the sub-region is defined

The four corners of the region ((Xi,Yi), i=1,4) defined by GETRG2 can be determined from the returned data in the following way:

```
X1 = X
Y1 = Y

X2 = X + REAL(NU - 1) * UX
Y2 = Y + REAL(NU - 1) * UY

X3 = X + REAL(NV - 1) * VX
Y3 = Y + REAL(NV - 1) * VY

X4 = X2 + X3 - X1
Y4 = Y2 + Y3 - Y1
```

Start and end layers are returned for the Z direction in LAY1 and LAY2. In the X and Y directions the data specifies a regular grid of NU by NV sampling points. If the grid positions are numbered from 1 to NU in the U direction and from 1 to NV in the V direction, the start position position of the subregion (X,Y) maps onto grid position (1,1), and the vectors (UX,UY) and (VX,VY) map onto the unit grid vectors (1,0) and (0,1).

GETRG2 returns .TRUE. only in case of error.

See also GETRG1.

The sub-region keys and options are:

```
SIZE, SI2
POSITION, PO2
LEFT/RIGHT
TOP/BOTTOM

SI3        }
PO3        }  or  LAYER, LA2
NEAR/FAR   }

SAMPLING
ANGLE/UV
```

If option UV is set, the Semper variables U, U2, V and V2 specify the vectors in picture coordinates which correspond to unit U and V grid vectors.

See section 6.5 of the Semper 6 Plus Advanced Users' Guide and the EXTRACT command for more details about these keys and options.

```
LOGICAL FUNCTION GETRNG(PMIN,PMAX,LPN)
```

Returns the minimum and maximum pixel values in the picture with logical picture number LPN.

This routine has been replaced by SEMRNG. A call to GETRNG can be replaced by the equivalent call to SEMRNG as follows:

```
      IF (SEMRNG(1,PMIN,PMAX,LPN)) RETURN
```

GETRNG has been retained in Semper's object library to maintain backwards compatibility.

```
LOGICAL FUNCTION INVFT2(NPIC,CLASS,MSQ,ZEROC,RNORM)
```

Effects 2-D inverse Fourier transform from logical picture LP1 to picture number NPIC, returning the logical picture number of the output picture in LP2. Options are provided for taking the modulus squared of the source picture and zeroing the central pixel prior to transformation and for normalising the result.

Argument	Type	I/O	
NPIC	I	I	fully specified output picture number
CLASS	I	I	output picture class
			CLASS = NCLIMA, Image = NCLMAC, Macro = NCLFOU, Fourier = NCLSPE, Spectrum = NCLCOR, Correlation = NCLWAL, Walsh = NCLPLI, Position list = NCLLUT, Display look-up table = NCLHIS, Histogram = NCLUND, Undefined
MSQ	L	I	MSQ = .TRUE., the modulus-squared of the data is taken prior to transformation
ZEROC	L	I	ZEROC = .TRUE., zero the central pixel prior to transformation
RNORM	R	I	determines normalisation for the output result
			RNORM < 0, the result is divided by final central value = 0, the result is divided by the number of points > 0, the result is multiplied by RNORM

COMMON variables

Variable	Type	I/O	
LP1	I	I	logical picture number assigned to a suitable source picture (size must be such that the size of the output picture is factorisable)
LP2	I	O	logical picture number assigned to the output picture

Row buffers

Buffer	Type	I/O
RBS		used as workspace

If the source picture has its origin at the centre of the left hand side, it is taken to represent the right-hand half-plane of the forward transform and the resulting inverse transform will be entirely real (ouptut picture form is floating-point rather than complex). Otherwise, the origin must be at the centre of the source picture and the source picture is taken to represent a full-plane forward transform.

The size of the output picture depends on whether the source picture treated as a full-plane or half-plane transform. The Y dimension of the output picture is always the same as the Y dimension of the source picture. If the source picture is a full-plane transform, the output picture will have the same X dimension as the source picture. If the source picture is a half-plane transform, the X dimension will be 2 * (NCOL - 1), where NCOL is the X dimension of the source picture.

The size of the source picture must be such that the dimensions of the output picture are factorisable (they must have factors 2, 3, 4 and 5 only and must be a multiples of 4). The Semper command SHOW SIZES lists all the factorisable sizes supported by INVFT2.

The output picture is assigned the class specified by CLASS - INVFT2 does not check the output picture class. It is the caller's responsibility to ensure that the class is appropriately specified. Some Semper commands will not work with a picture derived from an inverse Fourier transform if the picture does not have an appropriate form, or they may produce different results according to the picture class (see commands ACF, XCF, FOURIER and PS).

WARNING: INVFT2 uses all of the row buffer array RBS. Data held in any of the row buffers will not be preserved.

INVFT2 returns .TRUE. if an error is detected (e.g. size of the output picture is not factorisable, position of the source picture origin is incorrect).

See also FT1D and FT2D.

```
INTEGER FUNCTION IPACK(STRING)
```

Returns the packed integer representation for the alphanumneric string of characters in STRING.

Argument	Type	I/O	
STRING	C*(*)	I	string of alphanumeric characters ('A' to 'Z', 'a' to 'z', '0' to '9' and '$')

Only the first three characters in STRING are significant. Upper-case characters are treated as if they were lower-case characters and any invalid characters are treated as if they were blanks.

The following routines expect names in packed integer form:

- IVAL
- IVALPN
- OPT
- OPTNO
- SEMKTX
- SEMLU
- SEMTEX
- SETVAR
- UNPACK
- UNSETV
- VAL
- VARINT
- VARSET

See also UNPACK.

```
INTEGER FUNCTION IVAL(NAME)
```

Returns the current value of the Semper variable with packed name NAME as an integer.

Argument	Type	I/O	
NAME	I	I	packed name of Semper variable

NAME = IPACK('<variable name>')

IVAL returns the current integer value of the specified Semper variable.

IVAL is functionally equivalent to:

```
IF (VARSET(NAME)) THEN
   IVAL = NINT(VAL(NAME))
ELSE
   IVAL = 0
ENDIF
```

See also IVALPN and VAL.

```
INTEGER FUNCTION IVALPN(NAME)
```

Returns the current value of the Semper variable with packed name NAME as a valid picture number.

Argument	Type	I/O	
NAME	I	I	packed name of Semper variable

NAME = IPACK('<variable name>')

IVALPN returns the current value of the Semper variable as a fully specified picture number.

If the variable value does not include a device component the current value of the Semper variable CD is used.

IVALPN is functionally equivalent to:

```
IF (IVAL(NAME).LT.1000) THEN
   IVALPN = IVAL(NAME) + 1000 * IVAL(IPACK('cd'))
ELSE
   IVALPN = IVAL(NAME)
ENDIF
```

See also IVAL and VAL.

```
LOGICAL FUNCTION KLINE(PROMPT,ISSUE,STRING,NS)
```

Issues a prompt and reads a line of input from the keyboard with line editing and recall facilities. The routine returns when <Return> is pressed.

Argument	Type	I/O	
PROMPT	C*(*)	I	prompt string to be output at the start of the line
ISSUE	L	I	ISSUE = .TRUE., the prompt string is output = .FALSE., no prompt string is output
STRING	C*(*)	I	initial line of input text
		O	line of text input from the keyboard
NS	I	I	NS = 0, contents of STRING ignored > 0, STRING(1:NS) output as an initial line of input text
		O	NS = 0, blank line input > 0, STRING(1:NS) contains input text with trailing blanks removed

KLINE returns .FALSE. unless an error is detected (BREAK detected, problem accessing keyboard). Input, and line editing, is limited to LEN(STRING) characters.

The characters returned in STRING are restricted to the standard ASCII set. Any invalid key presses will be ignored and will cause the terminal bell to be sounded.

For a simple routine to wait for and return single key and mouse button presses see KPRESS.

LOGICAL FUNCTION KPRESS(KEY)

Returns a key code depending on the key or mouse button pressed. Mouse button presses are returned as special key codes. If no keyboard or mouse button input is pending, the routine waits for input, returning when input is detected.

Argument	Type	I/O	
KEY	I	O	next key in keyboard buffer, failing that the next mouse button pressed and released

KEY = key code, ordinary key pressed
= KBFUNC + <n>, function key <n> pressed
= KMBUT + <n>, mouse button <n> pressed

If a keyboard key is pressed, the code will lie in the range 0 to 127. If a function key is pressed, the code will lie in the range KBFUNC to KBFMAX. If a mouse button is pressed, the code will lie in the range KMBUT+1 to KMBMAX.

The full set of valid key codes is defined in the INCLUDE file called ICSET.

KPRESS reads and ignores all pointer events.

KPRESS returns .TRUE. in case of error (e.g. if BREAK is detected).

See also KLINE, KREAD.

LOGICAL FUNCTION KREAD(KEY)

Returns a key code depending on the key or mouse button pressed. Mouse button presses are returned as special key codes. KREAD does not wait for input if none is present and returns KBNONE if keyboard and button queues are empty. KREAD must be used in conjunction with EVOPEN and EVCLOS.

Argument	Type	I/O	
KEY	I	O	key code if keyboard queue not empty, failing that, mouse button code if button queue not empty KEY = KBNONE, no key or mouse button pressed = key code, ordinary key pressed = KBFUNC + <n>, function key <n> pressed = KMBUT + <n>, mouse button <n> pressed

If a keyboard key is pressed, the code will lie in the range 0 to 127. If a function key is pressed, the code will lie in the range KBFUNC to KBFMAX. If a mouse button is pressed, the code will be in the range KMBUT+1 to KMBMAX.

KREAD returns .FALSE. unless an error is detected (BREAK detected, EVOPEN not called before KREAD).

If a mouse button 'close' is detected, KREAD waits until either the button is opened or another relevant event occurs (keyboard input, if enabled, or BREAK).

The full set of valid key codes is defined in the INCLUDE file called ICSET.

If the keyboard and mouse button queues have not been activated then KREAD will return KBNONE for the key code.

KREAD reads and ignores all pointer events.

The event processing loop which involves KREAD should be bracketed by calls to EVOPEN (to activate the keyboard and/or button queues) and EVCLOS (to restore the state of the event queues).

For a self-contained routine to wait for and read a single key or mouse button press see KPRESS.

For a self-contained routine to wait for and read a line of text from the keyboard see KLINE.

See also EVFLUS, EVQENQ.

LOGICAL FUNCTION MARSET(ANNOT,MARK)

Reads the MARK key and determines if annotataion is required, in the standard way.

Argument	Type	I/O	
ANNOT	L	O	ANNOT = .TRUE., annotation is appropriate
MARK	I	O	display picture to be annotated

If the MARK key is set and its value is greater than zero, ANNOT is set to .TRUE. and MARK returns a valid display picture number. If the value of the MARK key is in the range 1 to 1000, the picture number is obtained from the value of Semper's DISPLAY variable. Calling MARSET reflects the standard arrangement whereby setting the MARK key to NO suppresses annotation and setting the MARK key to YES causes the current display picture to be annotated.

MARSET should be used in conjunction with FSINIT in the following manner:

```
IF(MARSET(ANNOT,MARK)) RETURN

IF(ANNOT) THEN
   IF(FSINIT(3,MARK)) RETURN

   IF(FSxxx( .... )) RETURN
ENDIF
```

MARSET returns .FALSE. unless an error occurs (MARK key set to invalid display picture number).

```
SUBROUTINE MCTIME(TIME)
```

Returns current date and time as an array of integer values.

Argument	Type	I/O	
TIME	I(7)	O	current date and time

```
TIME(1) = year (for example, 1994)
TIME(2) = 1 to 12, month
TIME(3) = 1 to 31, day
TIME(4) = 0 to 23, hours
TIME(5) = 0 to 59, minutes
TIME(6) = 0 to 59, seconds
TIME(7) = 0 to 99, centiseconds
```

Time may be measured in coarser units than centiseconds (0.01 second), in which case the number of centiseconds (TIME(7)) will be rounded to the nearest equivalent time quantum.

LOGICAL FUNCTION MEANSD(LPN)

Returns the range of pixel values and the mean and the standard deviation for logical picture LPN, or for a specified subregion of the picture. The results are returned in the Semper variables MEAN, ME2, SD, MIN and MAX.

Argument	Type	I/O	
LPN	I	I	logical picture number for the source picture

COMMON variables

Variable	Type	I/O	
SMGL1	L	I	SMGL1 = .FALSE., the whole picture is to be scanned = .TRUE., subregion to be scanned is defined by the limits in SMGI1 to SMGI6
SMGI1	I	I	first column of the picture included in the subregion
SMGI2	I	I	" row " " " " " " "
SMGI3	I	I	" layer " " " " " " "
SMGI4	I	I	last column of the picture included in the subregion
SMGI5	I	I	" row " " " " " " "
SMGI6	I	I	" layer " " " " " " "

Row buffers

Buffer	Type	I/O	
RB1			used as workspace

If SMGL1 is .FALSE., the whole of the source picture is scanned and the picture range is recorded in the picture label (the values of SMGI1 to SMGI6 are ignored). Otherwise, SMGI1 to SMGI6 define the limits of the subregion of the source picture to be scanned.

The minimum and maximum pixel values are returned in the Semper variables MIN and MAX. For complex pictures, MIN and MAX return the combined range of the real and imaginary parts.

The mean pixel value is returned in the Semper variable MEAN. If the picture contains complex data, separate means for the real and imaginary parts of the picture are returned in Semper variables MEAN and ME2.

The standard deviation is returned in the Semper variable SD. For complex pictures this is a combined estimate based on the deviation of the modulus about the mean.

The COMMON variables SMGL1 and SMGI1 to SMGI6 can be set to appropriate values by calling the routine GETRG1 immediately before calling MEANSD:

```
      IF (GETRG1(COL1,ROW1,LAY1,COL2,ROW2,LAY2,LPN)) RETURN

      SMGL1 = COL1 .NE. 1 .OR. COL2 .NE. NCOLS(LPN) .OR.
     +        ROW1 .NE. 1 .OR. ROW2 .NE. NROWS(LPN) .OR.
     +        LAY1 .NE. 1 .OR. LAY2 .NE. NLAYS(LPN)

      IF (SMGL1) THEN
         SMGI1 = COL1
         SMGI2 = ROW1
         SMGI3 = LAY1
         SMGI4 = COL2
         SMGI5 = ROW2
         SMGI6 = LAY2
      ENDIF

      IF (MEANSD(LPN)) RETURN
```

GETRG1 reads the subregion keys SIZE, POSITION and LAYER and options LEFT/RIGHT, TOP/BOTTOM and NEAR/FAR.

WARNING: Row buffer RB1 is used as workspace. Values held in RB1 will not be preserved.

MEANSD returns .TRUE. only in case of error (e.g. malformed picture label).

See also GENHST, RANGE and SEMRNG.

```
CHARACTER*1 FUNCTION MKCHAR(ICHA)
```

Returns the single character corresponding to integer ASCII code ICHA.

Argument	Type	I/O	
ICHA	I	I	integer ASCII code

MKCHAR returns the single character corresponding to ICHA. On your system, it may or may not be the same as the FORTRAN intrinsic function CHAR.

See also MKICHA and SEMCHS.

INTEGER FUNCTION MKICHA(CHA)

Returns the integer ASCII code corresponding to first character of CHA.

Argument	Type	I/O	
CHA	C*(*)	I	character string

MKICHA returns the integer ASCII code corresponding to CHA(1:1). On your system MKICHA may or may not be the same as the FORTRAN intrinsic function ICHAR.

See also MKCHAR and SEMICS.

```
INTEGER FUNCTION NBLANK(STRING)
```

Returns the position of the last non-blank character of STRING, or zero if all the characters in STRING are blanks.

Argument	Type	I/O	
STRING	C*(*)	I	character string

NBLANK returns the useful length of STRING, i.e. the position of the last non-blank character.

LOGICAL FUNCTION OBEYCL(STRING)

Executes the contents of STRING as a string of Semper commands when the current command terminates.

Argument	Type	I/O	
STRING	C*(*)	I	string containing one or more Semper commands

OBEYCL queues the string of Semper commands for execution after the current command terminates, for example

```
IF (OBEYCL('trap=-1 assign scratch size 100; if ~rc stop')) RETURN
```

causes a 100 Kbyte temporary picture disk to be assigned.

If several calls to OBEYCL are made in sequence, the commands will be executed in the order they were input, for example

```
IF(OBEYCL('type x')) RETURN
IF(OBEYCL('type y')) RETURN
```

is functionally equivalent to

```
IF(OBEYCL('type x; type y')) RETURN
```

The command interpreter reports errors occurring in commands queued by OBEYCL by reporting the usual error message with an additional message:

```
In immediate command
```

OBEYCL functions as if a call is made immediately after the current command has terminated to an un-named library procedure containing the commands specified in STRING.

OBEYCL returns .TRUE. if any error occurs.

WARNING: Beware of passing strings of commands to OBEYCL which could cause further calls to OBEYCL. The library call depth sets a limit on the number of calls to OBEYCL which can be nested in this way (use the command SHOW SYSTEM to find out what the library call depth is).

```
LOGICAL FUNCTION OPT(NAME)
```

Returns .TRUE. if the Semper variable with packed name NAME is set and has a non-zero value. You use OPT to test for Semper command options.

Argument	Type	I/O	
NAME	I	I	packed name of Semper option

NAME = IPACK('<option name>')

OPT is functionally equivalent to:

```
IF (VARSET(NAME)) THEN
   OPT = VAL(NAME) .NE. 0.0
ELSE
   OPT = .FALSE.
ENDIF
```

See also OPTNO.

```
LOGICAL FUNCTION OPTNO(NAME)
```

Returns .TRUE. if the Semper variable with packed name NAME is set and has a zero value. You use OPTNO to test for negative Semper command options, such as NOVERIFY and NOERASE.

Argument	Type	I/O	
NAME	I	I	packed name of Semper option
			NAME = IPACK('<option name>')

Note that NAME should be set to the packed value for the option name without the NO prefix, for example, IPACK('verify') and IPACK('erase') to test for NOVERIFY and NOERASE.

OPTNO is functionally equivalent to:

```
IF (VARSET(NAME)) THEN
   OPTNO = VAL(NAME) .EQ. 0.0
ELSE
   OPTNO = .FALSE.
ENDIF
```

See also OPT.

```
LOGICAL FUNCTION PDELTA(IDX,IDY,N,IEVENT,ICODE)
```

Reads the pointer event queue, recording all pointer movements, until the pointer event queue is emptied or the arrays IDX and IDY are full, or until a terminating event is encountered (key press or button open or close). PDELTA must be used in conjunction with EVOPEN and EVCLOS.

Argument	Type	I/O	
IDX	I(*)	O	array of X and Y pointer movements
IDY	I(*)	O	.
N	I	I	length of arrays IDX and IDY
		O	number of pointer movements returned
IEVENT	I	O	type of terminating event detected
			IEVENT = 0, no terminating event (pointer queue is empty or arrays IDX and IDY are full)
			= 1, key pressed
			= 2, button closed
			= 3, button opened
ICODE	I	O	if IEVENT = 1, key code
			= 2 or 3, mouse button number

PDELTA incorporates support for simulating pointer movement with the cursor keys on the keyboard, including the facility to increase or decrease the pointer step size by means of the 'C' or 'c' and 'F' or 'f' keys (coarse and fine). Use of the 'C', 'c', 'F', 'f' keys and the cursor keys does not terminate the operation of PDELTA.

The pointer movements in IDX and IDY are returned as unscaled integer values, unrelated to any particular coordinate system. It is the responsibility of the calling routine to scale these values appropriately. Positive values in IDX and IDY denote movement of the pointer respectively from left to right and upwards.

The full set of valid key codes is defined in the INCLUDE file called ICSET.

The processing loop which involves PDELTA must be bracketed by calls to EVOPEN (to activate the pointer keyboard and button queues) and EVCLOS (to restore the state of the event queues).

If the pointer queue is inactive no pointer movements will be returned.

PDELTA returns .TRUE. in case of error (e.g. break event detected, no prior call to EVOPEN, N less than 1).

See also PSIGMA, EVFLUS, EVQENQ.

```
LOGICAL FUNCTION PSIGMA(IDX,IDY,IEVENT,ICODE)
```

Reads the pointer event queue, summing all pointer movements, until the event queue is emptied or until a terminating event is encountered (key press or button open or close). PSIGMA must be used in conjunction with EVOPEN and EVCLOS.

Argument	Type	I/O	
IDX	I	O	total pointer movement in X and Y direction
IDY	I	O	.
IEVENT	I	O	type of terminating event detected
			IEVENT = 0, no terminating event (pointer queue is empty)
			= 1, key pressed
			= 2, button closed
			= 3, button opened
ICODE	I	O	if IEVENT = 1, key code
			= 2 or 3, mouse button number

PSIGMA incorporates support for simulating pointer movement with the cursor keys on the keyboard, including the facility to increase or decrease the pointer step size by means of the 'C' or 'c' and 'F' or 'f' keys (coarse and fine). Use of the 'C', 'c', 'F', 'f' keys and the cursor keys does not terminate the operation of PSIGMA.

The pointer movement in is returned as a pair of unscaled integer values, unrelated to any particular coordinate system. It is the responsibility of the calling routine to scale these values appropriately. Positive values in IDX and IDY denote movement of the pointer respectively from left to right and upwards.

The full set of valid key codes is defined in the INCLUDE file called ICSET.

The processing loop which involves PSIGMA must be bracketed by calls to EVOPEN (to activate the pointer keyboard and button queues) and EVCLOS (to restore the state of the event queues).

If the pointer queue is inactive no pointer movement will be returned.

PSIGMA returns .TRUE. in case of error (e.g. no preceding call to EVOPEN, BREAK detected).

See also PDELTA, EVFLUS, EVQENQ.

```
LOGICAL FUNCTION RANGE(IOP,LPN)
```

Returns the range of pixel values for the picture with logical picture number LPN, or for a subregion of the picture. RANGE can also be used to delete or write the range record in the picture label. The range is passed via the Semper variables MIN and MAX.

Argument	Type	I/O	
IOP	I	I	IOP = 1 return range values in Semper variables MIN and MAX (the picture is not scanned if the range is recorded in the picture label)
			= 2 return range values in Semper variables MIN and MAX (the picture is scanned unconditionally)
			= 3 delete the range values in the picture label
			= 4 record values of Semper variables MIN and MAX as new range values in the picture label
LPN	I	I	logical picture number for the source picture

COMMON variables

Variable	Type	I/O	
SMGL1	L	I	SMGL1 = .FALSE., the whole picture is to be scanned = .TRUE., subregion to be scanned is defined by the limits in SMGI1 to SMGI6
SMGI1	I	I	first column of the picture included in the subregion
SMGI2	I	I	" row " " " " " " "
SMGI3	I	I	" layer " " " " " " "
SMGI4	I	I	last column of the picture included in the subregion
SMGI5	I	I	" row " " " " " " "
SMGI6	I	I	" layer " " " " " " "

Row buffers

Buffer	Type	I/O	
RB1			used as workspace

Determining the picture range: IOP=1,2

The minimum and maximum pixel values are returned in the Semper variables MIN and MAX. For complex pictures, the ranges for the real and imaginary parts of the picture are combined.

If IOP = 1 and the range for the entire picture is required (SMGL1 = .FALSE.), the range is obtained from the picture label, provided that the information has already been recorded in the picture label. If not, the picture is scanned and the new range information is recorded in the picture label.

If IOP = 2, the picture is scanned unconditionally. If SMGL1 = .FALSE., the range for the entire picture is determined and the range record in the picture label is updated accordingly.

If an abandon request is detected during the scan, the range values for the partial scan are returned in MIN and MAX, and RANGE returns .FALSE. with COMMON variable ERROR = 4. In this case, the range record in the picture label is not updated.

If SMGL1 is .FALSE., the entire source picture is scanned. Otherwise, SMGI1 to SMGI6 define the limits of a subregion of the source picture to be scanned.

The COMMON variables SMGL1 and SMGI1 to SMGI6 can be set to appropriate values by calling the routine GETRG1 immediately before calling RANGE:

```
      IF (GETRG1(COL1,ROW1,LAY1,COL2,ROW2,LAY2,LPN)) RETURN

      SMGL1 = COL1 .NE. 1 .OR. COL2 .NE. NCOLS(LPN) .OR.
     +        ROW1 .NE. 1 .OR. ROW2 .NE. NROWS(LPN) .OR.
     +        LAY1 .NE. 1 .OR. LAY2 .NE. NLAYS(LPN)

      IF (SMGL1) THEN
         SMGI1 = COL1
         SMGI2 = ROW1
         SMGI3 = LAY1
         SMGI4 = COL2
         SMGI5 = ROW2
         SMGI6 = LAY2
      ENDIF

      IF (RANGE(IOP,LPN)) RETURN
```

GETRG1 reads the subregion keys SIZE, POSITION and LAYER and options LEFT/RIGHT, TOP/BOTTOM and NEAR/FAR.

Deleting a range record: IOP=3

The range record in the picture label is deleted. This is done automatically by Semper's command interpreter at the end of a command if any data is output to a picture by calling SEMROW. If any changes are made to picture data which would invalidate the range record without involving SEMROW, the range record must be deleted explicitly with a call to RANGE.

Writing a range record: IOP=4

The values of the Semper variables MIN and MAX are written into the range record of the picture label. RANGE can be called to update the range record in this way if, as a result of some processing, the new picture range is known (it is the caller's responsibility to ensure that the range values do in fact correspond to the picture's current data values). Recording the information in the picture label saves a subsequent scan through the picture data to establish the range when it is needed.

WARNING: Row buffer RB1 is used as workspace. Values held in RB1 will not be preserved.

WARNING: RANGE can return .FALSE. with ERROR = 4.

RANGE returns .TRUE. only in case of error (e.g. malformed picture label).

See also GENHST, MEANSD and SEMRNG.

```
LOGICAL FUNCTION SEMBEE( )
```

Generate an audible sound (beep) at the terminal.

SEMBEE returns .FALSE. if successful.

```
LOGICAL FUNCTION SEMBRK( )
```

Returns BREAK status and flushes the break queue.

SEMBRK returns .TRUE. if an error or abandon request (ERROR = 4) is detected.

A call to SEMBRK clears the break queue so that subsequent calls to SEMBRK will return .FALSE. until such time as a new abandon request is encountered.

```
LOGICAL FUNCTION SEMCEN(LPN,NCCOL,NCROW,NCLAY)
```

Records NCCOL, NCROW and NCLAY as the new coordinate origin of the picture with logical picture number LPN.

Argument	Type	I/O	
LPN	I	I	logical picture number
NCCOL	I	I	new column number for origin
NCROW	I	I	new row number for origin
NCLAY	I	I	new layer number for origin

Row buffers

Buffer	Type	I/O	
RB1			used as workspace

SEMCEN returns .TRUE. if unsuccessful (e.g. new origin lies outside picture).

WARNING: Row buffer RB1 is used workspace to hold the picture label. Values held in RB1 will not be preserved.

```
SUBROUTINE SEMCHS(STRING,ITEXT,N)
```

Converts array of integer ASCII codes into a character string.

Argument	Type	I/O	
STRING	C*(*)	O	character equivalent of ITEXT
ITEXT	I(*)	I	array of integer ASCII codes
N	I	I	number of characters to be converted
			N < LEN(STRING), STRING(N+1:LEN(STRING)) = ' '
			> LEN(STRING), excess characters are discarded

The character string is padded with blanks or truncated as appropriate. An input value of N = 0 is acceptable, in which case a blank string is returned.

See also SEMICS and MKCHAR.

```
LOGICAL FUNCTION SEMCLS(LPN)
```

Closes a picture given its logical picture number (marks the entry in the logical picture table as 'suitable for re-use' by SEMOPN).

Argument	Type	I/O	
LPN	I	I	logical picture number

When you call SEMOPN to open a picture, it decides for itself which entry in the logical picture table to re-use. It may chose an entry for a picture which you have opened and wish to continue accessing. In this case, SEMCLS allows you to control the situation by declaring which logical picture numbers may definitely be re-used. You only need to call SEMCLS if you expect to access more than NLPS pictures during the execution of a single command.

SEMCLS returns .FALSE. if successful.

```
LOGICAL FUNCTION SEMCON(STRING)
```

Outputs a line of text to the console output stream.

Argument	Type	I/O	
STRING	C*(*)	I	character string to be output

SEMCON returns .FALSE. when successful.

Trailing blanks are not output.

See also SEMDIA, SEMINP, SEMLOG, SEMMON and the Semper commands ECHO and PAGE.

```
LOGICAL FUNCTION SEMDEL(IOP,NUMBER)
```

Performs various picture deletion operations.

Argument	Type	I/O	
IOP	I	I	IOP = 1, delete a single picture, if it is not write-protected
			= 2, scan all picture discs and delete all temporary pictures
			= 3, write a new empty directory to device NUMBER (deletes all existing pictures)
			= 4, delete a picture even if the label is malformed and ignoring write-protection
NUMBER	I	I	if IOP = 1 or 4, fully specified picture number = 2, NUMBER is not used = 3, device number

Row buffers

Buffer	Type	I/O	
RB1			used as workspace
RB2			used as workspace

Deleting a picture does not involve overwriting any image data, but the picture directory and the logical picture table (if the picture is currently opened) are amended to prevent further access to the picture. The storage allocated to the picture is made available for creating new pictures.

A call to SEMDEL with IOP = 3 writes a new empty directory for picture device NUMBER, deleting all pictures on that device.

SEMDEL returns .TRUE. if an error occurs (e.g. invalid picture number, disc error, request to delete a protected picture with IOP = 1 or 2, attempt to access a picture with a malformed label with IOP not set to 4).

WARNING: Row buffers RB1 and RB2 are used as workspace. Values held in RB1 and RB2 will not be preserved.

WARNING: There is no facility to undo a call to SEMDEL. Use SEMDEL with care (especially with IOP = 3) and keep backup copies of important data.

```
LOGICAL FUNCTION SEMDIA(STRING,ISEVER)

Outputs a line of text to the diagnostic output stream.

Argument    Type      I/O

STRING      C*(*)     I      character string to be output

ISEVER      I         I      ISEVER = NDIMES, informational message
                                    = NDIWAR, warning message
                                    = NDIERR, error message
                                    = NDIFAT, fatal error message


All diagnostic messages should be output via this routine.  The parameters
for ISEVER are defined in the INCLUDE file called PARAMS.

SEMDIA returns .FALSE. unless an error is detected.

Trailing blanks are not output.

See also SEMCON, SEMINP, SEMLOG, SEMMON and the Semper commands ECHO and PAGE.
```

```
SUBROUTINE SEMICS(STRING,ITEXT,N)
```

Converts character string into array of integer ASCII codes.

Argument	Type	I/O	
STRING	C*(*)	I	character string
ITEXT	I(*)	O	array of integer ASCII codes
N	I	I	number of characters to be converted
			N < LEN(STRING), excess characters are discarded
			> LEN(STRING), ITEXT is padded with blanks

The character string is padded with blanks or truncated as appropriate. An input value of N = 0 is acceptable, in which case the routine does nothing.

See also MKICHA and SEMCHS.

```
LOGICAL FUNCTION SEMINP(STRING)
```

Outputs a line of text the logical output stream devoted to the input of non-command text.

Argument	Type	I/O	
STRING	C*(*)	I	character string to be output

All non-command input should be reflected by calling this routine, since this ensures that every detail of an interactive session can be logged.

SEMINP returns .FALSE. unless an error is detected.

Trailing blanks are not output.

See also SEMCON, SEMDIA, SEMLOG, SEMMON and the Semper commands ECHO and PAGE.

```
LOGICAL FUNCTION SEMINT(NOPROG)
```

Returns information about Semper's interactive state. It allows you to find out whether Semper is running in batch mode or when the UIF menu system is active, and also whether a Semper RUN file or LIBRARY program is running.

Argument	Type	I/O	
NOPROG	L	I	NOPROG = .FALSE., return .FALSE. if (1) Semper is running in batch mode or (2) UIF menu system is active = .TRUE., return .FALSE. if (1) Semper is running in batch mode or (2) UIF menu system is active or (3) RUN command is active or (4) LIBRARY command is active

A call to SEMINT allows you to check for certain combinations of the following states:

(1) Semper is running in batch mode (BATCH variable is set to 1)
(2) The UIF menu system is active (UIF variable is set to 1)
(3) Semper is currently executing a run file (RUN command is active)
(4) Semper is currently executing a program (LIBRARY command is active)

Note that you can check for the first two states directly by examining the protected variables BATCH and UIF (being protected variables, they are both guaranteed to be set).

You would use SEMINT in an IF statement to bracket sections of code which will cause some sort of interaction with the user:

```
IF ( SEMINT( NOPROG ) ) THEN
   .
   < Some form of interaction with the user >
   .
ENDIF
```

If NOPROG is set to .TRUE., SEMINT will only return .TRUE. if none of the above states is active, that is, Semper is communicating directly with the user via the command line interface and Semper's event queues. This means that any form of interaction with the user is possible. If SEMINT returns .FALSE., some forms of interaction may not work (in batch mode, any form of interaction is impossible and any attempt to use an interactive facility will either generate an error or leave Semper waiting indefinitely for events which will never be generated).

Setting NOPROG to .FALSE. allows SEMINT to return .TRUE. when Semper is executing a program or run file. In these circumstances, it may be appropriate to allow some interaction with the user, for example, prompting the user for file names or other information.

```
LOGICAL FUNCTION SEMKTX(NAME,PROMPT,ITEXT,N,UCASE)
```

Returns text associated with a Semper key with packed name NAME. The text is returned in integer ASCII form in array ITEXT. If a value for the key has not been specified on the command line and PROMPT is non-blank, a prompt is output, and the text is input from the keyboard.

Argument	Type	I/O	
NAME	I	I	packed name of Semper text key NAME = IPACK('<text key name>')
PROMPT	C*(*)	I	string containing prompt to be issued if key is unset PROMPT = ' ', no interactive input allowed
ITEXT	I(*)	O	array of integer ASCII codes representing the input text
N	I	I O	length of array ITEXT amount of input text N = 0, no input text > 0, number of characters returned in array ITEXT
UCASE	L	I	UCASE = .TRUE., text is converted to upper-case

SEMKTX returns .TRUE. in case of error, but when accepting input interactively (in response to the prompt) it will trap any format errors and re-try until suitable input is received or a BREAK is detected.

If PROMPT contains a blank string, no interactive input of text is allowed. If, in addition to this, the key has not been specified on the command line, SEMKTX returns with no input text (N = 0). Interactive input of text is limited to 80 characters, even if the size of the return array ITEXT is greater than this (N > 80).

Text is returned in the array ITEXT as a series of integer ASCII codes. The input string is truncated to N characters. Use SEMCHS to convert the resulting integer array into a Fortran character string.

The text key must be followed by a properly formatted textstring (a series of quoted strings and/or arbitrary numerical expressions separated by commas). If the textstring is incorrectly formatted, SEMKTX returns .TRUE. and reports the fault with Semper error 3 - "Bad value for <key name>".

See also SEMTEX.

LOGICAL FUNCTION SEMLAB(IOP,LABEL,LPN)

Provides access to the picture label for the picture with logical picture number LPN.

Argument	Type	I/O	
IOP	I	I	IOP = 1, read picture label into array LABEL = 2, write contents of the array LABEL to the picture label
LABEL	I(LNLAB)		array containing picture label data as unsigned byte values (0 to 255) in integer form
		O	if IOP = 1, existing picture label data
		I	= 2, new picture label data
LPN	I	I	logical picture number

Parameters are supplied in the INCLUDE file called PARAMS to facilitate access to the various components of the picture label. For example, the parameter LBPLTY specifies the location in the picture label of the type code for a position list picture, i.e. the type code is accessed via LABEL(LBPLTY).

SEMLAB checks the contents of the picture label to ensure that the data is less likely to cause problems when accessing the picture data. It checks for a malformed picture label (LABEL(1 to 6) must contain the integer ASCII codes for the string "Semper"), bad data parameters which would invalidate access to the picture data (incorrect size, data form, picture class and bad counts for the range and title strings) and incorrect values for the position of the picture origin and creation date and time. Note that a bad picture origin can be reset with the ORIGIN RESET command. A bad creation date/time can be safely ignored.

When reading a picture label (IOP = 1), SEMLAB returns error 52 for a malformed label and error 147 for bad data parameters. When writing a picture label (IOP = 2), SEMLAB returns error 149 for a malformed picture label and error 148 for bad data parameters. If the origin or creation date/time parameters are incorrect, SEMLAB outputs a warning message.

SEMLAB returns .TRUE. in case of error (e.g. attempt to write a tape label, malformed label, incorrect data in label).

WARNING: Great care should be taken to ensure that the data in array LABEL is correctly formatted before writing it (IOP = 2). Failure to do this can lead to serious loss of data.

See also SEMTIT, SEMOPN, SEMCEN, RANGE and SEMRNG.

Warning: may write to RECORD !!

```
LOGICAL FUNCTION SEMLOG(STRING)
```

Outputs a line of text to the log output stream.

Argument	Type	I/O	
STRING	C*(*)	I	character string to be output

The log output stream is intended for outputing text to a log file instead of the terminal (e.g. long listings of tabulated values).

SEMLOG returns .FALSE. unless an error is detected.

Trailing blanks are not output.

See also SEMCON, SEMDIA, SEMINP, SEMMON and the Semper commands ECHO and PAGE.

```
LOGICAL FUNCTION SEMLU(IOP,NAME,VALUE)

Performs general operations on the Semper variable with packed name NAME.

Argument    Type     I/O

IOP         I        I     IOP = -1, return the value of the variable

                                  if SEMLU = .TRUE.,  variable is set
                                                      VALUE = variable value
                                           = .FALSE., variable is unset
                                                      VALUE = 0.0

                           = 0,   unset the variable

                                  if SEMLU = .FALSE., variable has been unset
                                           = .TRUE.,  variable is protected
                                                      or already unset

                           = 1,   set the variable to the value in VALUE

                                  if SEMLU = .FALSE., variable has been set
                                           = .TRUE.,  variable is protected or
                                                      variable table is full

                           = 2,   locally set the variable to the value in
                                  VALUE (the original state of the variable
                                  is restored at the end of the current
                                  Semper command)

                                  if SEMLU = .FALSE., variable has been set
                                           = .TRUE.,  variable is protected or
                                                      variable table is full

                           = 3,   save the current state of the variable for
                                  restoration at the end of the current
                                  Semper command (this is equivalent to using
                                  Semper's LOCAL command)

                                  if SEMLU = .FALSE., variable has been saved
                                           = .TRUE.,  variable is protected

NAME        I        I     packed name of Semper variable

                           NAME = IPACK('<variable name>')

VALUE       R          O   if IOP = -1,     current value of the variable
            R        I             = 1 or 2, new value for the variable
                                   = 0 or 3, VALUE is not used
```

Most of the functioning of SEMLU is provided in a more convenient form with the routines IVAL and VAL (to get the value of a variable), SETVAR and UNSETV (to set and unset a variable), and VARSET (to see whether a variable is set). SEMLU does however allow you to set or unset a variable locally and have the value restored if the processing of a Semper command is abandoned.

```
LOGICAL FUNCTION SEMLUT(IOP,LUTN,MODE,LUTBUF)
```

Provides access to display look-up table data stored in Semper's work disc.

Argument	Type	I/O	
IOP	I	I	IOP = 1, read look-up table data from work disc = 2, write look-up table data to work disc
LUTN	I	I	look-up table number
MODE	I	I/O	look-up table type MODE = 1, monochrome look-up table = 2, false colour look-up table = 3, full colour look-up table
LUTBUF I(LUTLEN,3)		I/O	array containing look-up table data

The look-up table number LUTN must be in the range 1 to NLUTS. All look-up table data values must lie in the range 0 to LUTMAX.

SEMPER stores look-up table data in the display as well as on the work disc. Any changes brought about by a call to SEMLUT must be reflected in the display by making a corresponding call to FSLURW.

Monochrome look-up table data are accessed in the array LUTBUF in the following manner:

```
      DO 10 I=1,LUTLEN
         GREY(I)=LUTBUF(I,1)
   10 CONTINUE
```

False and full-colour look-up table data are accessed in the array LUTBUF as follows:

```
      DO 10 I=1,LUTLEN
         RED(I)   =LUTBUF(I,1)
         GREEN(I)=LUTBUF(I,2)
         BLUE(I)  =LUTBUF(I,3)
   10 CONTINUE
```

SEMLUT returns .TRUE. in case of error (e.g. illegal look-up table number)

```
LOGICAL FUNCTION SEMMED(DEVICE,MEDIUM)
```

Returns the medium type associated with a given device.

Argument	Type	I/O	
DEVICE	I	I	device number
MEDIUM	I	O	MEDIUM = MEDVM, memory = MEDDC, disc = MEDDS, display = MEDFL, ASCII text file (log file) = MEDTP, tape

The device number DEVICE must be in the range 1 to NDVS.

The parameters NDVS, MEDVM, MEDDC, MEDDS, MEDFL and MEDTP are defined in the INCLUDE file called PARAMS.

SEMMED returns .FALSE. except in case of error. SEMMED faults a bad device number (DEVICE is outside the range 1 to NDVS) with error 76. If the specified device is not assigned, SEMMED returns with error 34.

LOGICAL FUNCTION SEMMON(STRING,MODULE,ICHAN)

Outputs a line of text to the monitor output stream (see the MONITOR command).

Argument	Type	I/O	
STRING	C*(*)	I	character string to be output
MODULE	C*(*)	I	name of routine calling SEMMON
ICHAN	I	I	ICHAN = 1 to NMCHAN, monitor output channel number

Text for monitoring the progress of a processing command should be output using SEMMON. The output should be assigned to a monitor output channel ICHAN in the range 1 to NMCHAN (SEMMON does nothing if ICHAN is out of range). NMCHAN is defined as a parameter in the INCLUDE file called PARAMS.

Each monitor output channel can be turned on or off seperately with the MONITOR command. The name of the routine calling SEMMON should be passed as a character string (this is not currently used but should be included for future use).

No attempt should be made to output any monitor output unless the COMMON variable MONIT is .TRUE. This avoids the unnecessary overhead of calling SEMMON when all monitoring is turned off, e.g.

```
      SUBROUTINE MYSUB

      INCLUDE 'COMMON'

      ......

      IF (MONIT) THEN
         IF (SEMMON('my monitor text','MYSUB', 15)) RETURN
      ENDIF

      ......

      END
```

SEMMON returns .FALSE. unless an error is detected.

Trailing blanks are not output.

See also SEMCON, SEMDIA, SEMINP, SEMLOG and the Semper commands ECHO, MONITOR and PAGE.

```
LOGICAL FUNCTION SEMOPN(IOP,NPIC,NCOL,NROW,NLAY,ICLASS,IFORM,LPN)

Opens a new or existing picture for further access.  When opening a new
picture, default parameters can be obtained from a designated source picture.

Argument    Type      I/O

IOP           I       I     IOP = 1, open an existing picture
                                  = 2, open a new picture
                                  = 3, open a temporary picture on any writable
                                       picture storage device (memory or disc)

NPIC          I       I     if IOP = 1 or 2, fully specified picture number

NCOL          I             picture size
NROW          I             .
NLAY          I             .
. . .

                        O   if IOP = 1,       old picture size
                      I             = 2 or 3, new picture size

ICLASS        I             picture class

                            ICLASS = NCLIMA, Image
                                   = NCLFOU, Fourier
                                   = NCLSPE, Spectrum
                                   = NCLCOR, Correlation
                                   = NCLWAL, Walsh
                                   = NCLPLI, Position list
                                   = NCLLUT, Display look-up table
                                   = NCLHIS, Histogram
                                   = NCLMAC, Macro
                                   = NCLUND, Undefined

                        O   if IOP = 1,       old picture class
                      I             = 2 or 3, new picture class

IFORM         I             picture form

                            IFORM =  NFMBYT, byte
                                  =  NFMINT, integer
                                  =  NFMFP,  floating point
                                  =  NFMCOM, complex

                        O   if IOP = 1,       old picture form
                      I             = 2 or 3, new picture form

LPN           I         O   logical picture number assigned to the opened picture

                      I     if IOP = 2 or 3 only:

                            LPN = 0, origin centred, no title string, position
                                     list type = "list" (if ICLASS = NCLPLI)
                                ~= 0, logical picture number for existing picture
                                     providing origin, title and position list
                                     information for the new picture
```

Row buffers

Buffer	Type	I/O	
RB1			used as workspace
RB2			used as workspace

When opening an existing picture (IOP = 1), the picture size, class and form are returned in NCOL, NROW, NLAY, ICLASS and IFORM.

When opening a new picture (IOP = 2 or 3), the picture size, class and form is given by NCOL, NROW, NLAY, ICLASS and IFORM, and LPN must be set to zero or to a valid logical picture number returned by a previous call to SEMOPN. If LPN is not zero, it points to an existing picture from which the origin (if the picture size is the same), title and position list type is copied. Otherwise, the picture origin is centred, the picture has no title string and the position list type defaults to "list" for a position list (ICLASS = NCLPLI).

If a new picture is opened and it has exactly the same characteristics as an already-opened picture, SEMOPN will return the logical picture number for the already opened picture. This is to allow for pictures to be processed "in-place" (where the source and destination is the same). Any command which cannot process pictures in-place should fault this condition with error 59.

When opening a temporary picture (IOP = 3), NPIC is ignored and the picture is marked for deletion at the end of the current command. Storage for a temporary picture is allocated on the first writable picture storage device (memory or disc) with enough space for the picture.

SEMOPN returns a logical picture number LPN that points to the entry in the logical picture table (LP table) for the picture that has been opened. The LP table provides all the information required to gain access to the picture. The following information in the LP table may be examined directly in COMMON after SEMOPN has been called:

Picture device and number	DEVN(LPN), PICN(LPN)
Picture size	NCOLS(LPN), NROWS(LPN), NLAYS(LPN)
Position of origin	CCOLN(LPN), CROWN(LPN), CLAYN(LPN)
Class and data form	CLASSN(LPN), FORMN(LPN)
Display black and white levels	GSMIN(LPN), GSMAX(LPN)
Display sampling interval	PXSAM(LPN)

The maximum number of pictures which may be open at any one time is limited by the maximum number of picture control blocks. This is defined by the Fortran parameter NLPS. If an attempt is made to open more than this number of pictures, SEMOPN will re-use the least-recently used entry in the LP table, effectively denying access to the corresponding picture. The routine SEMCLS can be called before SEMOPN to provide direct control over the re-use of LP table entries.

If NPIC points to a display device, NPIC specifies the number of the display partition in which the picture is displayed. If the picture is larger than the partition, the picture data will be undersampled when written to the display by SEMROW. SEMROW will fault any attempt to read data from an undersampled display picture. The undersampling factor is recorded in PXSAM(LPN). SEMOPN obtains the black and white levels for a display picture directly from the Semper variables MIN and MAX and stores these values in GSMIN(LPN) and GSMAX(LPN).

SEMOPN responds to the general command line options ERASE and VIEW. If ERASE is set, SEMOPN erases the associated display partition. If VIEW is set, the viewing conditions are modified to bring the associated display partition into the centre of the display monitor (if the display port will allow this).

SEMOPN will not allow a new picture to be opened if the amount of data for a picture row would overflow a row buffer. A row buffer can contain up to LNBUF byte values, LNBUF/LNINT integer values, LNBUF/LNREAL floating-point values and LNBUF/LNCOMP complex values.

WARNING: SEMOPN uses the row buffers RB1 and RB2 as workspace. Values held in these buffers will not be preserved.

WARNING: SEMOPN may return the same logical picture number for source and destination pictures if the pictures have identical characteristics.

WARNING: SEMOPN calls FSINIT when opening a display picture, cancelling the effect of any previous call to FSINIT. The coordinate system established by the previous call to FSINIT will not be preserved.

SEMOPN returns .TRUE. in case of error.

```
INTEGER FUNCTION SEMPPN(NPIC)
```

Converts the picture number NPIC to a fully specified picture number.

Argument	Type	I/O	
NPIC	I	I	picture number, with or without the device component

If the variable value does not include a device component the current value of the Semper variable CD is used.

SEMPPN is functionally eqivalent to:

```
      IF (NPIC .LT. 1000) THEN
         SEMPPN = NPIC
      ELSE
         SEMPPN = NPIC + 1000 * IVAL(IPACK('cd'))
      ENDIF
```

See also IVALPN.

```
LOGICAL FUNCTION SEMRNG(IOP,PMIN,PMAX,LPN)
```

SEMRNG allows you to read, write or delete the range parameters of the picture with logical picture number LPN.

Argument	Type	I/O	
IOP	I	I	IOP = 1, return range parameters in PMIN and PMAX = 2, update range parameters according to PMIN and PMAX
PMIN	R		minimum and maximum pixel values for picture
PMAX	R		.
		O	if IOP = 1, current values for picture
		I	= 2, new values for picture
			if PMIN <= PMAX, store new values
			> PMAX, delete old values
LPN	I	I	logical picture number for the source picture

Row buffers

Buffer	Type	I/O	
RB1			used as workspace

If IOP is set to 1, SEMRNG returns the picture's current range values. If the range is not recorded in the picture label, the picture is scanned to establish the minimum and maximum pixel values and these are recorded in the picture label. For a complex picture, the ranges for the real and imaginary parts of the picture are combined.

If IOP is set to 2, the picture's range parameters in the picture label are updated. If PMIN and PMAX specify a valid range (PMIN <= PMAX), their values are stored in the picture label. It is the caller's responsibility to ensure that the range values do in fact correspond to the picture's current data values. If PMIN is greater than PMAX, the picture's range parameters are deleted. Any subsequent attempt to read the range parameters will cause the picture to be rescanned.

If IOP is not 1 or 2, SEMRNG does nothing.

WARNING: Row buffer RB1 is used as workspace. Values held in RB1 will not be preserved.

SEMRNG returns .TRUE. if an error is detected.

See also GENHST, MEANSD, and RANGE.

```
LOGICAL FUNCTION SEMROW(IOP,BUFFER,IFORM,IROW,LAYER,LPN)
```

Provides access to a single picture row.

Argument	Type	I/O	
IOP	I	I	IOP = 1, read row of data from picture = 2, write row of data to picture
BUFFER		I/O	array containing picture row data
	B(*) I(*) R(*) CP(*)		IFORM = NFMBYT, byte data = NFMINT, integer data = NFMFP, floating-point data = NFMCOM, complex data
IFORM	I	I	form of data contained in BUFFER IFORM = NFMBYT, byte data = NFMINT, integer data = NFMFP, floating-point data = NFMCOM, complex data
IROW	I	I	picture row number
LAYER	I	I	picture layer number
LPN	I	I	logical picture number for source/destination picture, established by a call to SEMOPN

The logical picture number LPN points to the entry in the logical picture table (LP table) for the picture to be accessed. Semper supports a total of NLPS picture control blocks, numbered from 1 upwards. The LP table provides all the information required by SEMROW to access the picture data. The logical picture number and the information in the LP table are set-up by calling SEMOPN. SEMROW faults a bad value for LPN (LPN outside the range 1 to NLPS) with error 165 and reports the failure to call SEMOPN before SEMROW with error 166. Any attempt to write to a write-protected picture will be faulted with error 41.

The following information in the LP table may be examined directly in COMMON:

Picture device and number	DEVN(LPN), PICN(LPN)
Picture size	NCOLS(LPN), NROWS(LPN), NLAYS(LPN)
Position of origin	CCOLN(LPN), CROWN(LPN), CLAYN(LPN)
Class and data form	CLASSN(LPN), FORMN(LPN)
Display black and white levels	GSMIN(LPN), GSMAX(LPN)
Display sampling interval	PXSAM(LPN)

Picture data is accessed as an array of NCOLS(LPN) data values by specifying the row and layer numbers with IROW and LAYER which must fall in the range 1 to NROWS(LPN) and 1 to NLAYS(LPN) respectively. SEMROW faults a bad row number with error 9 and a bad layer number with error 61.

IFORM specifies the required data form for BUFFER. If this differs from the data form for the picture (FORMN(LPN)), SEMROW will convert the data to the destination data form. Note that when writing data (IOP = 2) to a complex display picture or to a tape picture, SEMROW will convert the data in place, replacing the original data in BUFFER.

Suitable floating point buffer arrays are supplied in COMMON by means of the overall row buffer array RBS and the six individual row buffer arrays RB1 to RB6 which are eqivalenced to RBS:

```
REAL RBS(1-LNEDGE:LNBUF/LNREAL+LNEDGE,NNBUF)
COMMON /SEMBUF/ RBS

REAL RB1(LNBUF/LNREAL),RB2(LNBUF/LNREAL),RB3(LNBUF/LNREAL)
REAL RB4(LNBUF/LNREAL),RB5(LNBUF/LNREAL),RB6(LNBUF/LNREAL)

EQUIVALENCE (RB1,RBS(1,1)),(RB2,RBS(1,2)),(RB3,RBS(1,3))
EQUIVALENCE (RB4,RBS(1,4)),(RB5,RBS(1,5)),(RB6,RBS(1,6))
```

The size of each of the row buffers RB1 TO RB6 is LNBUF bytes or LNBUF/LNREAL floating-point values. Note that RBS allows you to access edge pixels RBS(1-LNEDGE to 0,IBUF) and RBS(NCOLS(LPN)+1 to NCOLS(LPN)+LNEDGE),IBUF) directly. LNEDGE is guaranteed to be at least 10. On systems where NNBUF > 6, RBS also allows you to have more than six row buffers (IBUF > 6). Fortran compilers should generate equally efficient code for references to RBS(ICOL,1) (where the second subscript is a constant) as to references to RB1(ICOL).

When accessing integer data, an INTEGER buffer array should be equivalenced to the REAL array. Complex data can be accessed by means of an equivalenced COMPLEX array or by accessing pairs of data values in a REAL array. For example, IB1 and CB1 can be equivalenced to RB1 in the following way:

```
INTEGER IB1(LNBUF/LNINT)
COMPLEX CB1(LNBUF/LNCOMP)
EQUIVALENCE (IB1,RB1,CB1)
```

The equivalence between CB1 and RB1 is such that:

```
REAL(CB1(I))  = RB1(2*I-1)
AIMAG(CB1(I)) = RB1(2*I)          where I = 1,LNBUF/LNCOMP
```

In order to process byte picture data it should be converted to integer form by setting IFORM = NFMINT so that the data values are accesible in Fortran. Note that SEMROW will fault any attempt to read data which would overflow the row buffer when converted to form IFORM, with error 47.

Data is stored in memory or on disc or tape in the form specified when the picture was opened (FORMN(LPN)). If the form of the data (IFORM) you pass to SEMROW is different, SEMROW converts the data to the appropriate form. You can avoid any data conversions in SEMROW by first converting the data yourself with a call to CFORM and then calling SEMROW with IFORM set to FORMN(LPN).

With display pictures the data are scaled so that values equal to GSMIN(LPN) are displayed with brightness determined by the first entry in the display look-up table (entry 1). Values equal to GSMAX(LPN) are displayed with intensity determined by the last entry in the display look-up table (entry LUTLEN). Values outside the range GSMIN(LPN) to GSMAX(LPN) are truncated appropriately. Values between GSMIN(LPN) and GSMAX(LPN) are converted using linear interpolation and rounded to the nearest look-up table entry. This rounding effect means that there may be a loss of precision when data written to a display picture are read back using SEMROW. Note that it is permissible for GSMIN(LPN) to exceed GSMAX(LPN) (negative contrast).

The maximum and minimum intensity levels GSMAX(LPN) and GSMIN(LPN) are set by SEMOPN and modified by the DISPLAY command (if the PRESET option is not set).

If a display picture is larger than its associated display partition, undersampling of the data is used. The undersampling factor is stored in PXSAM(LPN). SEMROW does not allow data to be read (IOP = 1) from an undersampled display picture and faults such a request with error 33.

A complex display picture is displayed as two separate images side-by-side. The left-hand image represents the real part and the right-hand image represents the imaginary part of the picture.

SEMROW returns the value .TRUE. if an error is detected (e.g. attempt to read from an undersampled display picture, attempt to write to a write-proteced picture, attempt to read data which will not fit in a row buffer after being converted or abandon request).

WARNING: SEMROW sometimes converts the contents of BUFFER when IOP = 2 and a form conversion is required.

See also FSROW.

LOGICAL FUNCTION SEMSEL(LPN)

Defines which picture is to be selected on exit from a command.

Argument	Type	I/O	
LPN	I	I	LPN = 0, the automatic mechanism for selecting a picture is disabled for the duration of the current command
			> 0, logical picture number for picture to be selected on exit from the current command

SEMSEL returns .TRUE. in case of error (LPN does not refer to a valid picture opened by the current command).

```
LOGICAL FUNCTION SEMSOP( )
```

Signals the start of a fresh page of output by resetting the count of the number of lines output to the terminal.

Lines of text output to the terminal are counted so that when the count approaches the maximum for the current terminal page size, a page prompt can be output if the page prompt is enabled (with the command PAGE PROMPT). The routine SEMTPS can be called to obtain the current terminal page size. Calling SEMSOP ensures that the page prompt will not appear until a whole page of text has been output.

SEMSOP should be called after an interruption resulting from any form of interactive input (e.g. command line input, XWIRES input, etc.). Note that SEMSOP is called automatically by the command interpreter whenever commands are input interactively and whenever routine KLINE is called.

SEMSOP returns .FALSE. unless an error occurs.

```
LOGICAL FUNCTION SEMTEX(NAME,ITEXT,N)
```

Returns text associated with a Semper key with packed name NAME. The text is returned in integer ASCII form in array ITEXT.

Argument	Type	I/O	
NAME	I	I	packed name of Semper text key NAME = IPACK('<text key name>')
ITEXT	I(*)	O	array of integer ASCII codes representing the associated textstring
N	I	I	length of array ITEXT
		O	amount of text N = 0, text key is not present > 0, number of characters returned in array ITEXT

SEMTEX looks for the command line text key specified by NAME. If none is found, N is returned set to zero. If the key is present, it must be followed by a properly formatted textstring (a series of quoted strings and/or arbitrary numerical expressions separated by commas). If the textstring is incorrectly formatted, SEMTEX returns .TRUE. and reports the fault with Semper error 3 - "Bad value for <key name>".

Text is returned in the array ITEXT as a series of integer ASCII codes. The input string is truncated to N characters. Use SEMCHS to convert the resulting integer array into a Fortran character string.

See also SEMKTX.

```
LOGICAL FUNCTION SEMTIT(IOP,TITLE,LPN)
```

Provides access to the title string for the picture with logical picture number LPN.

Argument	Type	I/O	
IOP	I	I	IOP = 1, read the title string = 2, write the title string
TITLE	C*(*)		title string for the picture
			TITLE = ' ', picture with no title string
		O	if IOP = 1, return picture title (the title string is truncated or padded with blanks as appropriate)
		I	= 2, output new picture title (characters beyond the maximum length for a title string and trailing blanks are ignored)
LPN	I	I	logical picture number

Parameters in the INCLUDE file called PARAMS define the maximum length of the title string as follows:

```
maximum length = LBTT2 - LBTT1 + 1
```

SEMTIT returns .FALSE. if successful.

```
LOGICAL FUNCTION SEMTPS(IWID,ILEN)
```

Returns the current terminal page size.

Argument	Type	I/O	
IWID	I	O	page width in characters
ILEN	I	O	page length in lines

The user-defined page size is the page size defined by the PAGE command. This, when truncated to the current hardware limits, results in the current terminal page size. The user-defined page size is most likely to be truncated on workstations where the terminal window size can be changed dynamically.

SEMTPS returns .FALSE. unless an error is detected.

LOGICAL FUNCTION SEMWAI(SECS)

Waits for a specified period of time, but checks at regular intervals for any abandon request.

Argument	Type	I/O	
SECS	R	I	wait time in seconds (up to 29000 seconds)

SEMWAI will try to wait for at least as long as specified, but the exact duration depends on the local implementation of Semper. Once the specified time has elapsed SEMWAI will return.

The maximum wait time is 29000 seconds (about 8 hours). Wait times greater than this are truncated to 29000 seconds.

SEMWAI returns .TRUE. if an error or abandon request (ERROR = 4) is detected.

See also WAITS, SEMBRK.

LOGICAL FUNCTION SETVAR(NAME,VALUE)

Sets a Semper variable to a given value.

Argument	Type	I/O	
NAME	I	I	packed name for the Semper variable
			NAME = IPACK('<variable name>')
VALUE	R	I	value to which the variable is to be set

SETVAR returns .TRUE. in case of error (e.g. the variable is protected or the variable table is full and a new variable is to be set).

See also UNSETV.

```
CHARACTER*8 FUNCTION TIMSTR(IHOUR,IMINUT,ISECON)
```

Returns an 8 character string containing the specified time in text form. If any argument is out of range, asterisks '*' are substituted for it in the return string.

Argument	Type	I/O	
IHOUR	I	I	hours, in range 0 to 23
IMINUT	I	I	minutes, in range 0 to 59
ISECON	I	I	seconds, in range 0 to 59

Output is of the form:

```
TIMSTR = '08:15:54'
```

See also MCTIME and DATSTR.

```
CHARACTER*3 FUNCTION UNPACK(NAME)
```

Unpacks a name in packed integer form and returns a three character, lower-case, string.

Argument	Type	I/O	
NAME	I	I	packed integer representation of a Semper name

UNPACK performs the opposite function to routine IPACK.

```
LOGICAL FUNCTION UNSETV(NAME)

Unsets a Semper variable.

Argument    Type      I/O

NAME        I         I      packed name for the Semper variable

                             NAME = IPACK('<variable name>')
```

UNSETV returns .TRUE. in case of error (e.g. NAME refers to a protected variable).

See also SETVAR.

```
REAL FUNCTION VAL(NAME)
```

Returns the current value of the Semper variable with packed name NAME as a floating point value.

Argument	Type	I/O	
NAME	I	I	packed name of Semper variable
			NAME = IPACK('<variable name>')

If the specified variable is unset, VAL returns zero.

See also IVAL and IVALPN.

```
LOGICAL FUNCTION VARINT(NAME)
```

Returns .TRUE. if the Semper variable with packed name NAME is an exact integer or is unset.

Argument	Type	I/O	
NAME	I	I	packed name of Semper variable
			NAME = IPACK('<variable name>')

VARINT is functionally equivalent to:

```
VARINT = VAL(NAME) .EQ. ANINT(VAL(NAME))
```

See also VARSET.

LOGICAL FUNCTION VARSET(NAME)

Returns .TRUE. if the Semper variable with packed name NAME is set.

Argument	Type	I/O	
NAME	I	I	packed name of Semper variable

NAME = IPACK('<variable name>')

Semper variables which are not fixed or protected may be set or unset. When a variable is unset it does not have a value defined for it. Note that some functions, such as IVAL and VAL, return a zero value for an unset variable.

See also IVAL, VAL, SETVAR, UNSETV and VARINT.

SUBROUTINE WAITS(SECS)

Waits for a specified number of seconds.

Argument	Type	I/O	
SECS	R	I	wait time in seconds

WAITS will try to wait for at least as long as specified, but the exact duration depends on the local implementation of Semper. Once the specified time has elapsed WAITS will return.

WAITS cannot be interrupted by a BREAK or abandon request, and should not therefore be used when a long wait is required (more than 1 second). For a long wait use SEMWAI instead.

Chapter 4

FILE FORMATS

Introduction

This chapter provides detailed descriptions of the following file formats:

- READ/WRITE file format
- INPUT/OUTPUT file format

These file formats may be used to import images into Semper. They can also be used to transfer images and data between Semper systems running on different host computers.

Files which conform to the READ/WRITE file format are accessed with Semper's **read** and **write** commands. Such files are written using Fortran formatted or unformatted WRITE statements. Formatted files can be transferred from one Semper installation to another, but they tend to be large and slow to access. Unformatted files provide a more efficient way to transfer data between Semper and other Fortran programs, but the data may not be readable on different computer installations because of possible differences in Fortran file structure, byte ordering and floating-point format. Note also that when inputting an unformatted file with the **read** command, the **unformatted** option must be specified, otherwise **read** will assume that it is to read a Fortran formatted file.

Files which conform to the INPUT/OUTPUT file format are accessed with Semper's **input** and **output** commands with no format options specified. The data is written as a stream of bytes with a fixed byte ordering and standardised floating-point format. This file format provides both an efficient and portable means for transferring data between different computer installations.

READ/WRITE file format

The details for the READ/WRITE file format are presented in the form of example Fortran coding. Note that the type declaration INTEGER on most systems means INTEGER*2 (the command **show system** will list the number of bytes/integer for your version of Semper).

A picture file can be broken down into four parts:

- header record
- title record
- label record
- picture data

The header record and the picture data are always present in a picture file. The title record is present if the picture has a title. The label record is optional. The value of IFLAG specifies whether either record is present in the picture file via the variables ILABEL and NTITLE.

```
IFLAG = 10000*IVERSN + 1000*ILABEL + NTITLE
```

It also records the format in which the title record is stored and whether byte unformatted data is stored in byte or integer form via the variable the IVERSN. The **write** command omits the label record if the option **unlabelled** is specified.

The label record contains all of the data that makes up the picture label. Some of this information is duplicated in the header record, which is why the **read** command can manage without the label record: it constructs a picture label using the information in the header record.

The header record contains the picture size, class, data form, title string length and label record flag. For a Fortran formatted file, it also includes a format string for reading the pixel data. The header record is read or written in the following way, according to whether the file is formatted or unformatted:

Formatted READ:

```
READ (IUNIT,'(6I6,1X,A20)') NCOL,NROW,NLAY,ICLASS,IFORM,IFLAG,FORMAT
```

Formatted WRITE:

```
WRITE (IUNIT,'(6I6,1X,A20)') NCOL,NROW,NLAY,ICLASS,IFORM,IFLAG,FORMAT
```

Unformatted READ:

```
READ (IUNIT) NCOL,NROW,NLAY,ICLASS,IFORM,IFLAG
```

Unformatted WRITE:

```
WRITE (IUNIT) NCOL,NROW,NLAY,ICLASS,IFORM,IFLAG
```

The title record contains a string of up to 156 characters representing the picture title. This record is not present if the title string length NTITLE is zero. The form in which the characters are stored depends on the value of IVERSN. If IVERSN is zero, the string is stored as an array of integer Hollerith values. Otherwise, the string is stored in Fortran 77 character format.

Formatted READ:

```
IF (IVERSN.EQ.0) THEN
   READ (IUNIT,'(80A1)') (ITITLE(I),I=1,NTITLE)
ELSE
   WRITE (IUNIT,'(80A1)') (TITLE(I:I),I=1,NTITLE)
ENDIF
```

Formatted WRITE:

```
IF (IVERSN.EQ.0) THEN
   READ (IUNIT,'(80A1)') (ITITLE(I),I=1,NTITLE)
ELSE
   WRITE (IUNIT,'(80A1)') (TITLE(I:I),I=1,NTITLE)
ENDIF
```

Unformatted READ:

```
IF (IVERSN.EQ.0) THEN
   READ (IUNIT) (ITITLE(I),I=1,NTITLE)
ELSE
   READ (IUNIT) TITLE(1:NTITLE)
ENDIF
```

Unformatted WRITE:

```
IF (IVERSN.EQ.0) THEN
   WRITE (IUNIT) (ITITLE(I),I=1,NTITLE)
ELSE
   WRITE (IUNIT) TITLE(1:NTITLE)
ENDIF
```

The label record contains 256 integer values representing the contents of the Semper picture label. This record is not present if the label record flag ILABEL is set to zero.

Formatted READ:

```
READ (IUNIT,'(16I4)') (LABEL(I),I=1,256)
```

Formatted WRITE:

```
WRITE (IUNIT,'(16I4)') (LABEL(I),I=1,256)
```

Unformatted READ:

```
READ (IUNIT) (LABEL(I),I=1,256)
```

Unformatted WRITE:

```
WRITE (IUNIT) (LABEL(I),I=1,256)
```

The remaining data in the file records the pixel values for the picture. The number of picture columns must not exceed the maximum row length for a given Semper installation (the Semper command **show system** will display this information). The number of picture rows and layers is limited by the space available in Semper's picture disc. Pixel data in byte form are usually recorded as integer values, unless the file is unformatted and IVERSN is set to 2, in which case the data are recorded with two byte values packed as integer values. Pixel data in integer form are recorded as integer values and pixel data in floating-point or complex form are recorded as floating-point values.

Data in byte form (IFORM = 0):

If unformatted file and IVERSN = 2:

```
READ/WRITE ( ..... ) (BPIXEL(I),I=1,(NCOL+(LNINT-1))/LNINT)
```

Note: the array BPIXEL contains the data as a string of bytes

Otherwise:

```
READ/WRITE ( ..... ) (IPIXEL(I),I=1,NCOL)
```

Data in integer form (IFORM = 1):

```
READ/WRITE ( ..... ) (IPIXEL(I),I=1,NCOL)
```

Data in floating-point form (IFORM = 2):

```
READ/WRITE ( ..... ) (RPIXEL(I),I=1,NCOL)
```

Data in complex form (IFORM = 3):

Either:

```
READ/WRITE ( ..... ) (RPIXEL(I),I=1,2*NCOL)
```

or:

```
READ/WRITE ( ..... ) (CPIXEL(I),I=1,NCOL)
```

When writing pixel data in Fortran formatted form, care must be taken to ensure that a suitable format is used.

The format string may be specified to the **write** command by means of the **format** key, for example:

```
write name ' .... ' format '(1X,10F8.2)'
```

If no format string is specified, the following default format strings are use

Data in byte form (IFORM = 0):

```
FORMAT = '(1X,24I3)'
```

Data in integer form (IFORM = 1):

```
FORMAT = '(1X,12I6)'
```

Data in floating-point or complex form (IFORM = 2 or 3):

```
FORMAT = '(1X,1P6E12.5)'
```

The pixel values are recorded in the file with the picture row data arranged in the following way:

Formatted READ:

```
    DO 200 K=1,NLAY
       DO 100 J=1,NROW
          READ (IUNIT,FORMAT) ( ......... )
100    CONTINUE
200 CONTINUE
```

Formatted WRITE:

```
    DO 200 K=1,NLAY
       DO 100 J=1,NROW
          WRITE (IUNIT,FORMAT) ( ......... )
100    CONTINUE
200 CONTINUE
```

Unformatted READ:

```
    DO 200 K=1,NLAY
       DO 100 J=1,NROW
          READ (IUNIT) ( ......... )
100    CONTINUE
200 CONTINUE
```

Unformatted WRITE:

```
    DO 200 K=1,NLAY
       DO 100 J=1,NROW
          WRITE (IUNIT) ( ......... )
100    CONTINUE
200 CONTINUE
```

The Fortran variables and arrays used in the coding examples above are defined as follows:

```
INTEGER IUNIT,LNINT
INTEGER NCOL,NROW,NLAY
INTEGER ICLASS,IFORM
INTEGER IFLAG,NTITLE,ILABEL,IVERSN
INTEGER ITITLE(156),LABEL(256)
INTEGER ICCOLN,ICROWN,ICLAYN
INTEGER IWP
INTEGER IYEAR,IMONTH,IDAY
INTEGER IHOUR,IMINUT,ISEC
INTEGER NCRANG
INTEGER IPLTYP
CHARACTER*20 FORMAT
CHARACTER*156 TITLE
INTEGER BPIXEL( .... )
INTEGER IPIXEL( .... )
REAL    RPIXEL( .... )
COMPLEX CPIXEL( .... )
EQUIVALENCE (RPIXEL,CPIXEL)
```

`IUNIT` Fortran unit for accessing picture file

`LNINT` number of bytes/integer
= 2, if INTEGER means INTEGER*2
= 4, if INTEGER means INTEGER*4

`NCOL` number of picture columns
`NROW` number of picture rows
`NLAY` number of picture layers

`ICLASS` picture class
= 1, image
= 2, macro
= 3, fourier
= 4, spectrum
= 5, correlation
= 6, undefined
= 7, walsh
= 8, histogram
= 10, display look-up table

`IFORM` picture data form
= 0, byte
= 1, integer
= 2, floating-point
= 3, complex

`NTITLE` number of characters in picture title
= 0, title record not present in picture file

ILABEL label record flag
= 0, label record not present in picture file
= 1, label record present in picture file

IVERSN version number
= 0, title stored in Hollerith form
= 1, title stored in character form
= 2, title stored in character form, byte unformatted data stored in byte form

IFLAG picture title and label flag
= 10000*IVERSN + 1000*ILABEL + NTITLE

ITITLE array of 256 integer values making up the picture label title string (not used if NTITLE = 0)

LABEL array of 256 integer values making up the picture label (not used if ILABEL = 0)

ICOLN column number of picture orgin
ICROWN row number of picture orgin
ICLAYN layer number of picture orgin

IWP write protect flag
= 0, picture is not write–protected
= 1, picture is write–protected

IYEAR creation year of picture
= 1900 to 2155

IMONTH creation month of picture
= 1 to 12

IDAY creation day of picture
= 1 to 31

IHOUR creation hour of picture
= 0 to 23

IMINUT creation minute of picture
= 0 to 59

ISEC creation second of picture
= 0 to 59

NCRANG number of characters encoding the range of pixel values in the picture data
= 0, picture range is not recorded

IPLTYP position list type
= 0, picture is not a position list
= 1, position list
= 2, open curve
= 3, closed curve

`FORMAT` character variable containing the format string used to write the picture in a formatted picture file

If complete information about a picture is desired, the data in the picture label must be arranged in the following way:

`LABEL(1 to 6)` integer ASCII codes for the string 'Semper'
= 83, 101, 109, 112, 101 and 114 always

```
NCOL = 256*LABEL(7)  + LABEL(8)
NROW = 256*LABEL(9)  + LABEL(10)
NLAY = 256*LABEL(11) + LABEL(12)

ICCOLN = 256*LABEL(13) + LABEL(14)
ICROWN = 256*LABEL(15) + LABEL(16)
ICLAYN = 256*LABEL(17) + LABEL(18)

ICLASS = LABEL(19)

IFORM = LABEL(20)

IWP = LABEL(21)

IYEAR  = LABEL(22) + 1900
IMONTH = LABEL(23)
IDAY   = LABEL(24)

IHOUR  = LABEL(25)
IMINUT = LABEL(26)
ISEC   = LABEL(27)

NCRANG = LABEL(28)
```

`LABEL(29 to NCRANG+28)` integer ASCII codes for string encoding the picture range as two floating-point decimal strings separated by a comma

`LABEL(NCRANG+29 to 55)` not used (set to zero)

```
IPLTYP = LABEL(56)
```

`LABEL(101 to NTITLE+100)` integer ASCII codes making up the picture title string

`LABEL(NTITLE+101 to 256)` not used (set to zero)

The information in the picture label **must** agree with the parameters in the file header, otherwise it may lead to severe problems when the picture file is read by the **read** command (it could corrupt the picture device in which the picture is stored). Make sure that the following applies:

```
LABEL(7)  = NCOL / 256
LABEL(8)  = MOD(NCOL,256)
LABEL(9)  = NROW / 256
LABEL(10) = MOD(NROW,256)
LABEL(11) = NLAY / 256
LABEL(12) = MOD(NLAY,256)

LABEL(19) = ICLASS

LABEL(20) = IFORM
```

INPUT/OUTPUT file format

The details of the INPUT/OUTPUT file format are presented as C code examples.

A picture file can be broken down into four parts:

- file header
- title string
- picture label
- image data

The file header and image data are always present in a picture file. The title string is present if the picture has a title. The picture label is optional. The file header records the dimensions, class and data form of the picture and a flag which determines the length of the title sting and whether the title string and/or picture label are present in the picture file.

```
/*  File header definition  */

    typedef struct { short ncol;   /*  number of columns   */
                     short nrow;   /*  number of rows      */
                     short nlay;   /*  number of layers    */
                     short class;  /*  picture class       */
                     short form;   /*  data form           */
                     short flag;   /*  title + label flag  */
                   } file_header;
```

The contents of the picture file can be read in the following way:

```
#define NFMBYT 0
#define NFMINT 1
#define NFMFP  2
#define NFMCOM 3

    void read_data ( int, void *, int, int );

    int             fd, label, ntitle;
    file_header     header;
    short           title [ 156 ];
    picture_label   label;

/*  open picture file  */

    fd = open ( "my_file.pic", 0 );

/*  read file header  */

    read_data ( fd, &header, 6, NFMINT );

/*  extract picture label and title flags  */

    ilabel = header.flag / 1000;  /*  label flag                */
    ntitle = header.flag % 1000;  /*  length of title string    */

/*  read title string (if any)  */

    if ( ntitle ) read_data ( fd, title, ntitle, NFMINT );

/*  read picture label (if any)  */

    if ( ilabel ) read_data ( fd, &label, 256, NFMINT );

/*  read image data  */

    for ( k = 1; k <= header.nlay; k++ )
    {
        for ( j = 1; j <= header.nrow; j++ )
        {

/*  read row of pixels (row = j, layer = k)  */

            read_data ( fd, buffer, header.ncol, header.form );
        }
    }

/*  Close picture file  */

    close ( fd );
```

The file header, picture title string and picture label are stored in the picture file as a series of short integer values (2 bytes each). The image data is stored as unsigned byte data, signed short integer data (2 bytes per value), floating-point data (4 bytes per value) or complex data (8 bytes per value), according to the data form parameter recorded in the file header. Floating-point data must be stored in the IEEE floating-point format. A complex data value is stored as two floating-point values with the real part followed by the imaginary part. All data must be stored in the picture file with the least-significant byte of each data value stored first. If the host machine's byte ordering is different, the bytes must be swapped as appropriate.

```
void read_data ( int fd, void *buffer, int n, int form )
{
    short word = 1;
    int   size_of [ 4 ] = { sizeof ( unsigned char ),
                            sizeof ( short ),
                            sizeof ( float ),
                            sizeof ( float ) * 2
                          };

/*  Read data  */

    read ( fd, buffer, n * size_of [ form ] );

/*  See if host's byte ordering is not least-significant byte first  */

    if ( ! *( (unsigned char *) (&word) ) )
    {

/*  Swap bytes according to data form  */

        switch ( form )
        {
            case NFMBYT: break;  /*  do nothing  */
            case NFMINT: swap_short ( buffer, n     ); break;
            case NFMFP : swap_long  ( buffer, n     ); break;
            case NFMCOM: swap_long  ( buffer, n * 2 ); break;
        }
    }
}
```

```
void swap_short ( void *buffer, int n )
{
    unsigned char *bytes, temp;
    int i;

    bytes = (unsigned char *) buffer;

    for ( i = 0; i < n; i++, bytes += 2 )
    {
        temp     = bytes [ 0 ];
        bytes [ 0 ] = bytes [ 1 ];
        bytes [ 1 ] = temp;
    }
}

void swap_long ( void *buffer, int n )
{
    unsigned char *bytes, temp;
    int i;

    bytes = (unsigned char *) buffer;

    for ( i = 0; i < n; i++, bytes += 4 )
    {
        temp     = bytes [ 0 ];
        bytes [ 0 ] = bytes [ 3 ];
        bytes [ 3 ] = temp;

        temp     = bytes [ 1 ];
        bytes [ 1 ] = bytes [ 2 ];
        bytes [ 2 ] = temp;
    }
}
```

The title string is encoded as an array of short integer ASCII codes, up to a maximum of 156 characters.

The picture label is the data structure Semper maintains to record all the parameters associated with a picture. It does not have to be present in the file because the **Input** command can provide default values for all the missing parameters. It consists of 256 short integer values. The picture label is stored by Semper as an array of unsigned byte values, so all the values in the picture label are restricted to the range 0 to 255.

```
/*  Picture label definition  /*

    typedef struct { short magic [ 6 ];       /*  label header string     */
                     short ncol1, ncol2;      /*  no. picture columns     */
                     short nrow1, nrow2;      /*  no. picture rows        */
                     short nlay1, nlay2;      /*  no. picture layers      */
                     short ccol1, cco12;      /*  centre column number    */
                     short crow1, crow2;      /*  centre row number       */
                     short clay1, clay2;      /*  centre layer number     */
                     short class;             /*  picture class           */
                     short form;              /*  data form               */
                     short wp;                /*  write-protected flag    */
                     short dyear, month, day;     /*  creation date       */
                     short hour, minute, second;  /*  creation time       */
                     short nrange;            /*  length of range string  */
                     short range [ 27 ];      /*  range string            */
                     short pltype;            /*  position list type      */
                     short reserved [ 43 ];   /*  not used                */
                     short ntitle;            /*  length of title string  */
                     short title [ 156 ];     /*  title string            */
                   } picture_label;
```

The data in the picture label is interpreted in the following way:

```
label.magic [ 1 ] =  83, 'S'
label.magic [ 2 ] = 101, 'e'
label.magic [ 3 ] = 109, 'm'
label.magic [ 4 ] = 112, 'p'
label.magic [ 5 ] = 101, 'e'
label.magic [ 6 ] = 114, 'r'

ncol = 256 * label.ncol1 + label.ncol2
nrow = 256 * label.nrow1 + label.nrow2
nlay = 256 * label.nlay1 + label.nlay2

ccol = 256 * label.ccol1 + label.ccol2 = 1 to ncol
crow = 256 * label.crow1 + label.crow2 = 1 to nrow
clay = 256 * label.clay1 + label.clay2 = 1 to nlay

label.class =  1, image
            =  2, macro
            =  3, fourier
            =  4, spectrum
            =  5, correlation
            =  6, undefined
            =  7, walsh
            =  8, position list
            =  9, histogram
            = 10, display look-up table
```

```
label.form = NFMBYT = 0, byte
           = NFMINT = 1, integer
           = NFMFP  = 2, floating-point
           = NFMCOM = 3, complex

label.wp = 0, picture is not write-protected
         = 1, picture is write-protected

year = 1900 + label.dyear = 1900 to 2155

label.month  = 1 to 12
label.day    = 1 to 31

label.hour   = 0 to 23
label.minute = 0 to 59
label.second = 0 to 59

label.nrange = 0, range values are not recorded
             = 1 to 27, length of range string in array label.range[ ]

label.pltype = 0, picture is not a position list
             = 1, position list
             = 2, open curve
             = 3, closed curve

label.ntitle = 0, the picture does not have a title
             = 1 to 156, length of title string in array label.title[ ]
```

The information in the picture label **must** agree with parameters in the file header, otherwise it may lead to severe problems when the picture file is read by the **Input** command (it could corrupt the picture device in which the picture is stored). Make sure that the following is true:

```
label.ncol1 == header.ncol / 256
label.ncol2 == header.ncol % 256
label.nrow1 == header.nrow / 256
label.nrow2 == header.nrow % 256
label.nlay1 == header.nlay / 256
label.nlay2 == header.nlay % 256

label.class == header.class

label.form  == header.form
```

The header string for the picture label, the range string and the title string are each represented as an array of short integer ASCII codes. The header string must contain the string 'Semper', otherwise the picture label will be faulted with Semper error 52 'Picture . . . has a malformed label'.

If the range string is present, it records the range of the image data in character form as a pair of signed floating-point numbers (possibly with an exponent), separated by a comma. If the range string is converted into a zero-terminated character string, it can be decoded in the following way:

```
char  *range;
float minimum, maximum;
int   i;

range = (char *) malloc ( label.nrange + 1 );

for ( i = 0; i < label.nrange; i++ ) range [ i ] = label.range [ i ];

range [ label.nrange ] = '\0';

sscanf ( range, "%f,%f", &minimum, &maximum );

free ( range );
```

Note that the dimensions of the picture must not exceed the capacity of the Semper installation which is to read the file. The number of picture columns must not exceed the maximum row length (the command **show system** will display this information). The number of picture rows and layers are limited by the space available on the destination picture storage device.

Chapter 5

RESERVED NAMES

Introduction

This chapter contains a list of all the Semper routine names used in all the platforms on which Semper currently runs. Users wishing to write portable Semper code should avoid using any of these names, even if they are not used on their particular system.

The routine names are listed in alphabetical order. Long names are truncated to 8 characters and have an asterisk * appended to them. There is a separate list at the end of this chapter which lists all the long names in full and in alphabetical order. For example,

```
ALLOCATE_BUFFER
ALLOCATE_MEMORY
ALLOCATE_SEGMENT
```

in the list of long names, appear as a single entry

```
ALLOCATE*
```

in the main list of reserved names.

Reserved names
==============

$EIKCHA	ANIDKO	ASSPRP	BMBCHA	CCACHE	COLFT3	DEVDSH	DMANU
$EIKDEL	ANIDWL	ASSSCR	BMBCLP	CDISC	COMPM	DEVDSI	DONONE
$EIKEXI	ANIEND	ASSSIZ	BMBCOD	CDISRD	CONGET	DEVHOR	DOPCLO
$EIKOPE	ANIFPP	ASSTAP	BMBCTL	CENTRE_V*	CONHAN	DEVTEX	DOUPAK
A1CONV	ANIFRA	ATSLTO	BMBDAD	CFGCLN	CONOPT	DEVVER	DRAG
ABANDN	ANIGFF	ATSTXT	BMBDEL	CFGCMP	CONVERT_*	DFSCLR	DRAG2
ABMATH	ANIGKO	ATSWTO	BMBDIN	CFGERF	COPYP	DFSCMO	DRAWING_*
ABREAK	ANIIKO	AUTO_CEN*	BMBDIS	CFGINI	COPY_MEM*	DFSCOF	DRAW_3D_*
ACF	ANIINC	BACK_PLA*	BMBFLD	CFGUPP	CORECT	DFSCON	DRAW_ARC
ACINI	ANIM8	BACK_SEC*	BMBFRA	CFGVAL	COVAR	DFSCQU	DRAW_BAN*
ACTLUT	ANIMKO	BAKPRO	BMBFRE	CFORM	CRDWRT	DFSCRE	DRAW_BOR*
ADCINT	ANIMOU	BAND_COA*	BMBINC	CFPCOM	CREATE_C*	DFSCSA	DRAW_CIR*
ADHALT	ANIMTE	BAS2	BMBMSZ	CGETCWD	CREATE_F*	DFSDSH	DRAW_COR*
ADINIT	ANIOKO	BASFIT	BMBOLP	CGETENV	CREATE_S*	DFSDSI	DRAW_CUR*
ADINLN	ANIPSZ	BBCALLBA*	BMBRBK	CHABIT	CROUT	DFSDST	DRAW_MEN*
ADLOAD	ANIRBL	BBCHEK	BMBSYT	CHAMAG	CSEMTOU	DFSHOR	DRAW_MID*
ADLPRG	ANIRKO	BBGETCL	BMBWBK	CHARACTE*	CTF	DFSTEX	DRAW_MOD*
ADOULN	ANISEQ	BBGO	BMBWIP	CHECK_FO*	CTF1	DFSVER	DRAW_NOT*
ADRINT	ANISHW	BBINP	BMBYUV	CHISQ	CTF2	DFSXDC	DRAW_PLA*
ADRMEM	ANISKO	BBINTR	BMLUT	CHKEND	CTYPE	DFTM	DRAW_REC*
ADRUN	ANISRV	BBLOOK	BMMAP	CHKSD	CURSOR_O*	DFTM1	DRAW_SHA*
ADSINT	ANISTP	BBMESS	BMMAP2	CHKTRP	CURSOR_X	DFTM2	DRAW_SID*
ADTPST	ANISTT	BBMESSF	BMMAP3	CHULL	CURSOR_X*	DFTRD	DRAW_SLI*
ADVERR	ANISVA	BBREAD	BMMDSK	CINCOM	CURSOR_Y	DFTWR	DRAW_STR*
ADWADD	ANIVCD	BBREADS	BMMMEM	CLEAR_WI*	CURSOR_Y*	DFTXB2	DRAW_WIN*
ADWADI	ANIVKO	BBSEND	BMPDSK	CLIP	CURSOR_Z	DHOCLR	DRWCUR
ADWCTL	ANIWBL	BBSTAT	BMPMEM	CLIPPING*	CURSOR_Z*	DHOCMO	DRWRBB
ADWMEM	ANIWDR	BBSTOP	BMPROC	CLIP_X_M*	CUT	DHOCOF	DSFAIL
AFPDIV	ANIWFF	BBTYPE	BMVAPE	CLIP_Y_M*	C_HAND	DHOCON	DSKIZ
AFPERR	APPACK	BCLEAR	BMVRBB	CLIP_Z_M*	D$HAND	DHOCQU	DSKIZ1
AFPINV	APPACT	BCOUNT	BMVWBB	CLITYP	DACGET	DHOCRE	DSKIZ2
AFPOVF	APPBEL	BDIFF	BMVWCB	CLLRCM	DACPUT	DHOCSA	DSPSUB
ALLOCATE*	APPGET	BDILER	BOPCLO	CLLSRS	DATE	DHODSH	DT
ALLOCM	APPSET	BEEP	BORDER_O*	CLOFIL	DATRAN	DHODSI	DT2
ALLOCX	APPSTR	BFBYT	BOX	CLOSED	DATSTR	DHODST	DTCLTR
ALTKEY	APPVAL	BFCOM	BPRINT	CLOSE_DI*	DAVAIL	DHOHOR	DTCLTW
ALTZONE	APPWAI	BFFPT	BPSEND	CLOSE_FI*	DAYLIGHT	DHOSCR	DTFRME
AMBIENT_*	APTEST	BFILL	BREP	CLOSE_ME*	DCDVIE	DHOTEX	DTHR1
ANALY2	APTINP	BFINT	BRKAST	CLOSE_WI*	DDILER	DHOVER	DTHR2
ANALY3	AQUERY	BFORM	BSCAN	CLOSFC	DEASS	DHOXCL	DTILTR
ANALY4	AREA42	BGNDOP	BSET	CLOST	DEASS2	DHOXGC	DTILTW
ANALY5	ASK	BHMT	BSHIFT	CLPERA	DEBUG	DHOXOP	DTMLTR
ANALY6	ASPECT_R*	BIOMES	BUFENDTA*	CLRBIT	DECUNP	DHOXPC	DTMLTW
ANALY7	ASSDIS	BIOPIX	BUZZER	CLS	DELETE_F*	DIFFER	DTOCW
ANALY8	ASSFIL	BIOTXS	BVALUE	CLSOVP	DESTRP	DISC	DTRAN1
ANALYS	ASSFS	BITCHA	CACHE	CLSRCM	DETER	DISP	DTRAN2
ANIAKO	ASSHDR	BITS	CACHECOL*	CLSVDU	DEVCLR	DISPLAY_*	DTRAN3
ANIANN	ASSIG	BLDRGL	CALC	CMATCH	DEVCMO	DLABL1	DTRAN4
ANIBEG	ASSIG2	BLKMEA	CALXVR	CNDNAM	DEVCOF	DLABL2	DTREAD
ANIBKO	ASSMEM	BLKVAR	CAMERA	CNVHEX	DEVCON	DLABL3	DTWRTE
ANIBLK	ASSNAM	BLOGIC	CAPAC2	CNVVAL	DEVCQU	DLABL4	DUPLICAT*
ANIDBL	ASSNEW	BMAP33	CAPACT	COLFT	DEVCRE	DLABL5	DVCHK
ANIDIR	ASSOLD	BMBADR	CAPTIO	COLFT2	DEVCSA	DLABL6	DZONE

ECDEC
ECHO
EDGE
EDIT2
EDIT4
EDITCALL*
EDITOR
EDSTUB
EIGSRT
EIKBYA
EIKCHA
EIKCLO
EIKDEL
EIKEXI
EIKINT
EIKLIN
EIKOPE
ELEVATIO*
ENDERR
ENSSTK
ENVINI
EPOL
EQB
EQFLUS
EQGETD
EQINIT
EQK
EQNEXT
EQNEXX
EQNQRE
EQNQRX
EQP
EQREAD
EQS
EQSETD
EQSETS
EQSETX
EQTERM
EQXBCL
EQXBDA
EQXBIN
EQXBOP
EQXBPL
EQXBRR
EQXFSO
EQXKCL
EQXKDA
EQXKER
EQXKIN
EQXKOP
EQXKPL
EQXPCL
EQXPDA
EQXPFL
EQXPIN
EQXPOP
EQXPOR
EQXPPL
EQXSCL
EQXSDA
EQXSFL
EQXSIN
EQXSOP
EQXSPL
EQXSWR
EQXXBC
EQXXBE
EQXXBO
EQXXCC
EQXXCL
EQXXKC
EQXXKE
EQXXKI
EQXXKO
EQXXLP
EQXXMM
EQXXMS
EQXXPB
EQXXPC
EQXXPE
EQXXPK
EQXXPL
EQXXPM
EQXXPO
EQXXPP
EQXXPS
EQXXPV
EQXXQE
EQXXQF
EQXXQR
EQXXQT
EQXXSC
EQXXSE
EQXXSL
EQXXSO
EQXXST
EQXXWI
EQXXWR
EQXXZB
ERASE
ERDWRT
ERRHAN
ERRNO
ERROR
ERRORREP
ESTARE
EVAL
EVCLOS
EVENT2
EVENTC
EVFLUS
EVOPEN
EVQENQ
EXA2
EXA3
EXA4
EXA5
EXAMPD
EXDONE
EXDRAW
EXIT
EXPAND
EXTIME
EXTRA2
EXTRA3
EXTRA4
EXTRAC
EXTRAP
EXTRCB
EXTRCT
EXTRNN
EYE_AZIM*
EYE_DIST*
EYE_ELEV*
EYE_X
EYE_Y
EYE_Z
F$MODE
F2ICNV
F6CHKB
F6CHKP
F6CHKT
F6INIT
F6OCHK
F6ODAT
F6ODO
F6OINS
F6OJMP
F6ONXT
F6OOD
F6OSET
F6OSTA
F6OSTR
F6OWAI
F6OXFR
F6PCHK
F6PDO
F6PJMP
F6PNXT
F6POD
F6PSET
F6PSKP
F6PSTA
F6PWAI
F6PXFR
F6UCHK
F6UDO
F6UINS
F6UJMP
F6UNXT
F6UOD
F6USET
F6USTA
F6UWAI
F6UXFR
FEATUR
FFT
FG100
FGETENV
FGETPAT
FGILUT
FGNDOP
FGRAB
FGVIDE
FIEAXS
FIEFLL
FIGA
FIGAGT
FIGB
FIGBOX
FIGBP
FIGC
FIGCC
FIGCD
FIGCHI
FIGCJ
FIGCN
FIGCND
FIGCS
FIGCW
FIGCX
FIGDLN
FIGE
FIGF
FIGFIG
FIGG
FIGH
FIGHMN
FIGILU
FIGINF
FIGINI
FIGL
FIGLTC
FIGLTI
FIGLTR
FIGLTW
FIGM
FIGMSG
FIGO
FIGP
FIGS
FIGSB
FIGSP
FIGST
FIGST2
FIGSTR
FIGSYN
FIGT
FIGT2
FIGT3
FIGTXT
FIGU
FIGV
FIGVER
FIGVID
FIGVO
FIGW
FIGX
FIGXWI
FILCDI
FILCIO
FILCLS
FILDCD
FILDIR
FILELM
FILEXI
FILKTX
FILL_IMA*
FILL_MEM*
FILL_OVE*
FILMAK
FILOPN
FILPAT
FILPOS
FILSEA
FILSTR
FIND
FINDCM
FIR33
FIR55
FIRF2
FIRFIL
FIRN1
FIRPEP
FITLC
FITLC2
FITSBD
FLOOD
FLOOD2
FLUCLI
FLUFIL
FLUSH
FLUSHD
FLUSHDIS*
FLUSH_DE*
FLUSH_SE*
FNDDIR
FNDFIL
FNDLSQ
FNDSIZ
FORCE_UP*
FORM_TO_*
FORTDE
FORTRD
FORTWR
FORURD
FORUWR
FOV_ANGL*
FOV_MOVE*
FPEXIT
FRDWRF
FRDWRT
FRDWRU
FREEDISP*
FREEFRAM*
FREEM
FREE_CAC*
FREE_DOS*
FREE_MEM*
FREE_SEG*
FREMSE
FRONT_PL*
FRONT_SE*
FROWSW
FSAMPL
FSARC
FSARRO
FSAS61
FSASGA
FSBAND
FSBFSL
FSBLCK
FSBNRI
FSBNRO
FSBORD
FSBRBL
FSBSOF
FSBSRI
FSBSRO
FSBUSY
FSCC61
FSCINI
FSCINT
FSCIRC
FSCLI
FSCLIP
FSCNRI
FSCSRI
FSCT61
FSCTRL
FSCTYP
FSCURS
FSCURV
FSDE61
FSDLN
FSDPLY
FSDRAG
FSEDGE
FSELEC
FSER61
FSERAS
FSERR
FSEX61
FSEXTG
FSFL61
FSFLL
FSFLUS
FSFNRI
FSFNRO
FSFRME
FSFSRI
FSFSRO
FSGR61
FSGRAB
FSHALT
FSIN61
FSINIT
FSINLN
FSINRI
FSINRO
FSINUB
FSISCA
FSISRI
FSISRO
FSLIHL
FSLINE
FSLIST
FSLN61
FSLQ61
FSLTR
FSLTSL
FSLTW
FSLU61
FSLURW
FSMAPB
FSMARK
FSMMBK
FSMSK
FSMSKF
FSOCW
FSOFF
FSOI61
FSOL61
FSOO61
FSOPTN
FSOQ61
FSOS61
FSOSCA
FSOULB
FSOULN
FSOUUB
FSOV61
FSOVRW
FSPR61
FSPRGM
FSPSOF
FSQ
FSQBOR
FSQCHA
FSQLIM
FSQLIV
FSQMON
FSQTRA
FSRB61
FSRDOV
FSREGN
FSRI61
FSRO61
FSROW
FSROWI
FSRUIS
FSRUN
FSSAMB
FSSAMI
FSSEI
FSSEND
FSSINT
FSSTAT
FSSYNC
FSTEST
FSTEXT
FSTX61
FSUO61
FSV01D
FSV02D
FSV03D
FSV04D
FSVAK1
FSVAK2
FSVAK3
FSVAK4
FSVAS0
FSVDE0
FSVDIN
FSVDMK
FSVERA
FSVERF
FSVERS
FSVERX
FSVGA
FSVIEW
FSVLU1
FSVLU2
FSVLU3
FSVLU4
FSVQ61
FSVRD1
FSVRD2

FSVRD3	GAMMLN	GGTOLP	HFILL	INTERRUP*	ITECAC	LEN_TRIM	LMANUF
FSVRD4	GAU	GID_TO_W*	HFILL2	INTFE	ITECAM	LETTER	LMANUR
FSVRDO	GBLINI	GLASS	HIDE_CLI*	INTPC	ITECAQ	LFSINT	LOAD_SEG*
FSVRO1	GBLRLW	GMCAMERA	HIDE_LIG*	INTRUP	ITECAS	LFSTRP	LOCALS
FSVRO2	GBLRWD	GMCAMERA*	HILBRT	INTWAI	ITECBC	LIBMAIN	LOCKFL
FSVRO3	GBLWLW	GMCLOSE	HISTLOCK	INVALOP	ITECCD	LIGHT_1_*	LOD1
FSVRO4	GBLWWD	GMCLOSEF	HISTMAX	INVER1	ITECFG	LIGHT_2_*	LOD2
FSVROV	GCBACK	GMDONE	HISTOG	INVERR	ITECFS	LIGHT_3_*	LOD3
FSVW61	GCFRA	GMDONEF	HISTPOS	INVERT	ITECH	LIGHT_BL*	LOD4
FSVWD1	GCHKFL	GMENQUIR*	HLCLIP	INVFT2	ITECHM	LIGHT_EL*	LONG
FSVWD2	GCONB	GMERASE	HODPRP	INVUVP	ITECIN	LIGHT_GR*	LOOKAT_X
FSVWD3	GCONI	GMFILL	HOLMES	INWORD	ITECLA	LIGHT_RE*	LOOKAT_Y
FSVWD4	GCYRGS	GMFILTER	HOURGLAS*	IOB	ITECLN	LIGHT_RO*	LOOKAT_Z
FSVWDO	GDATE	GMFILTER*	HPBFLU	IOBR62	ITECLP	LINDRW	LPDCEN
FSVWO1	GDRWCR	GMGETDEF*	HPBPUT	IOBW62	ITECLR	LINE_3D_*	LPDCLR
FSVWO2	GENHS2	GMINIT	HPBWRT	IOCTL	ITECLS	LINE_TO_*	LPDER
FSVWO3	GENHST	GMLINE	HPLJ	IOFILTER	ITECLU	LIVBEG	LPDET
FSVWO4	GENTM1	GMLIST	HPLJFD	IOINIT	ITECMR	LIVBKO	LPDSH1
FSVWOV	GENXY	GMLIVE	HSVRGB	IOINTR	ITECMS	LIVBLK	LPTPUT
FSWADD	GETCL	GMLUTVIE*	HWREG_LO*	IOMR1W	ITECOV	LIVCOP	LREAD
FSWADI	GETDCC	GMOPEN	I2FCNV	IOMRNW	ITECPB	LIVDBL	LSHIFT
FSWCTL	GETDISPL*	GMOPENF	IBSWAP	IOMW1W	ITECPC	LIVDEL	LSQERR
FSWDAT	GETDRV	GMOUTLUT	ICLIP	IOMWNC	ITECPX	LIVDKO	LSQFIT
FSWMEM	GETENV	GMOUTLUT*	ICNUM	IOMWNW	ITECRC	LIVE	LSQFT1
FSWNDW	GETEST	GMOVERLA*	IFQUERY	IOMX1W	ITECRF	LIVEND	LSTCPY
FSWT61	GETFIL	GMOVERRE*	IFSRME	IOMXNC	ITECSD	LIVEON	LSTFIR
FSWTXT	GETFIN	GMOVERVI*	IG3MSG	IOREAD	ITECSU	LIVGKO	LSTHST
FSX11M	GETFLG	GMOVERWR*	ILUT	IORR1W	ITECSZ	LIVIOP	LSTMAG
FSX11P	GETFRM	GMPOS	IMAACC	IORRNW	ITECTX	LIVMKO	LSTMAP
FSXBOX	GETHID	GMREAD	IMAGE	IORRXC	ITECVA	LIVMOU	LSTORE
FSXCM	GETMSE	GMREADF	IMARAN	IORRXW	ITECVB	LIVOKO	LSTSTT
FSXCTE	GETNAM	GMSETDEF*	IMGCLO	IORW1W	ITECVI	LIVOOP	LT1I
FSXDCU	GETNXT	GMSETGAI*	IMGDEL	IORWNC	ITECVQ	LIVPIC	LT1O
FSXL61	GETPAT	GMSETOFF*	IMGERR	IORWNW	ITECVW	LIVPKO	LT2I
FSXLST	GETRG1	GMSNAP	IMGNAM	IORX1W	ITECXY	LIVRBL	LT2O
FSXS61	GETRG2	GMSNAPF	IMGOPE	IOSMSG	ITECZP	LIVREG	LUNCLI
FSXSCC	GETRNG	GMSTANDA*	IMGRNG	IOTERM	IVAL	LIVSEQ	LUNSER
FSXSOC	GETSCA	GMSUPPOR*	IMGROW	IOWR62	IVALPN	LIVSHW	LUTAD2
FSXSOO	GETSDIR	GMSYNC	INARR1	IOWRIT	JAC	LIVSKO	LUTADA
FSXW61	GETT	GMSYNCF	INARR2	IOWW62	JEOL	LIVSLV	LUTADJ
FSXWIR	GETTMP	GMWRITE	INBYTE	IP12_BAS*	JEOLC	LIVSMP	LUTADK
FSXWPH	GETWDIR	GMWRITEF	INICAM	IP4_BASE	JUMP	LIVSOP	LUTADL
FSXWTR	GETXWE	GRAB	INITFC	IP5_BASE	KBDAST	LIVSSV	LUTADM
FSXWVF	GET_BAND*	GRAB_DOS*	INITM	IP6_BASE	KBDTRA	LIVSTP	LUTADO
FSXWVW	GET_BASE*	GRADX	INKEY	IPACK	KEEPINVE*	LIVSTT	LUTADV
FSZZI0	GET_CHAR*	GRADY	INOSP	IPTGET	KLINE	LIVTKO	LUTSET
FT1	GET_COLO*	GRADY1	INPCUT	IP_NUMBE*	KLINEW	LIVUKO	LZWUNP
FT1D	GET_DISP*	GRAPH	INPDUM	IQRNGE	KMOVE	LIVWAI	MACRO
FT1D2	GET_FONT*	GSERIS	INPLNK	IRDWRT	KPRESS	LIVWBL	MAGSR
FT1D3	GET_IMAG*	GSIGHT	INPPAI	IROWSW	KRDWRT	LIVWFR	MAGSUB
FT1D4	GET_INTE*	GTIME	INPPCX	ISCDRD	KREAD	LIVWKO	MAIN
FT1D5	GET_OVER*	GZINIT	INPRAS	ISLOCC	LABEL	LMANU	MAIN_
FT2D	GET_POS	GZOOM	INPRAW	ISLOCK	LABEL2	LMANU2	MAKE_FIL*
FTLMSQ	GET_SHAD*	HALFUL	INPUNF	ISLOCS	LABEL3	LMANU3	MAPPIX
FTSMAN	GET_VERT*	HANDLE_M*	INPUTF	ISSDRD	LABEL4	LMANU4	MARK
GAM	GFTOPL	HEADSW	INTERPOL*	IST	LATTIC	LMANU5	MARK2

MARMSG MLTIWD OUTCUT PFILTR PSIGMA RDEDXP RS2ON SECTN
MARPUT MNERR OUTDUM PHIST PSORT RDEMC RS2OPR SECTN2
MARSET MNOTON OUTPCX PHOTO PSOUT RDEXYD RS2RCV SECTN3
MASK MODEL_CO* OUTPTF PHRAND PSOUT0 RDHTCN RS2XMT SECURECA*
MASK1 MONITM OUTPUT_P* PICTUR PSOUT1 RDMODE RSCERR SELCAM
MASK2 MONITORL* OUTRAS PID PSOUT2 RDPRES RSGET SELDSK
MASK3 MONOTO OUTRAW PINDEX PSOUT3 RDSCC RSHIFT SELECT_C*
MAXHOD MORPH OUTTIF PIPCNT PSOUT4 RDSTIG RSPUT SELECT_Q*
MAXIMUM_* MOTIF OUTUNF PIPFLL PSOUT5 REACAM RSRBYT SELECT_R*
MAXLIK MOUSE OUWORD PIPGAI PSOUT6 REACFG RSRCHA SELECT_S*
MAXX MOUSE_TO* OVEACC PIPGRB PSOUTB REACLI RSRNUM SEMAHIST
MAXY MOVE OVEFL PIPINP PSOUTC READLN RSRSTR SEMAIN
MAXZ MOVELN OVERLY PIPLTW PTREAD READPM RSSBYT SEMAINFO
MAX_BAND* MOVE_SLI* OVREAD PIPMSK PTYPE READSI RSSEND SEMALIST
MAX_MODE* MRDBIN OVWRIT PIPOCW PUSHBT READ_DEV* RSTIME SEMARG
MCDBIN MRDGRV PAGE PIPOFF PUTFIL READ_FIL* RSUM1D SEMASK
MCDC61 MRDRFF PAKHEX PIPSUP PUTFIN READ_IMA* RSWAIT SEMBEE
MCDC61S MRDRFI PAKINT PIPSYN PUTFLG READ_MEM* S6BMOV SEMBRK
MCDCUT MRKREG PARTIT PIPTVS PUTOUT READ_OVE* S6BNRO SEMBUF
MCDGET MRTRA2 PASTE PMANU PWRGEN READ_SEG* S6BSRO SEMCBS
MCDOPN MRTRAN PATH PMARK PXCON REAFE S6DISP SEMCEN
MCDPCX MRWBIN PCA PNTAST PXSCA REAFIL S6ERAS SEMCFG
MCDUMP MSPACE PCACAL PNXAST QGET REALUT S6FACT SEMCHS
MCICUT MWRBIN PCALC PORDER QLTW REDRAW_D* S6FILT SEMCIP
MCIDUM MYPOS PCF170 POSITION* QPUT REDRAW_I* S6FNRO SEMCLS
MCKBIN NBLANK PCFACC PQUERY QRDWRT REDRAW_M* S6FRAM SEMCOM
MCOCUT NOCHAR PCFCLT PRDWR2 QSHADE REDRAW_S* S6FSRO SEMCON
MCODUM NOGOOD PCFCNT PRDWRT QTYPE REGNAM S6GROW SEMCP2
MCQBIN NOISE PCFDSO PRINT2 QUERY REINI2 S6HDOV SEMCPW
MCTIM2 NOISE2 PCFFBA PRINTC QUERYCAL* REINIT S6HDPI SEMCTX
MCTIME NRESET PCFFLL PROJEC QUERY_CA* REPORT S6HSPI SEMDBI
MCTP61 NXTLIN PCFGRB PROMPT QUERY_DE* RESIZE_D* S6IMAP SEMDCR
MEANSD OBEYCL PCFINT PRTHEL QUERY_FI* RESIZE_F* S6INRO SEMDDB
MEMBLK OCF PCFLTO PRUADD QUERY_ME* RESIZE_M* S6ISRO SEMDEL
MENTOU OCF2 PCFLTW PRUCAS QUERY_SE* RESTORE_* S6LIVE SEMDER
MENVIS OFQUERY PCFMLT PRUCOM QUEST RETUR S6LOAD SEMDIA
MERGE OKAY PCFOFF PRUCOP RABBIT REWIN S6MAKE SEMDIN
MGLASS OPEFIL PCFWTO PRUDIR RADX50 RFILTR S6OWEM SEMDLN
MGLBEG OPEN_DIS* PCONB PRUENT RAMPS RGB S6RDCD SEMDPD
MGLCPY OPEN_FIL* PCUAST PRUFLS RANGE RGBHSV S6RDOV SEMDPN
MGLDBG OPEN_MEM* PCURV2 PRUIND RANK RGBIN0 S6SYNC SEMDRO
MGLEND OPEN_NEW* PCURVE PRUINF RBOD32 RGBINT S6VECC SEMECH
MGLFG OPEN_OLD* PCXAST PRULAB RBSWAP RGBLU0 S6VECS SEMEND
MGLINI OPEN_SCR* PDELTA PRUPRI RCCHEK RGBLUT S6VIEW SEMENV
MGLPCH OPEN_WIN* PDRAW PRUSHO RCF RGBMON S6WORK SEMEOL
MGLROG OPNRCM PEAKS PRUSLT RCF2 ROUND_AN* S6WRK0 SEMERR
MGLZCH OPT PEAKS2 PRUSOR RCMCL RQUERY S6WRKI SEMEVE
MINCLA OPTNO PEAKS3 PRUTIN RCMERR RS232 S6WROV SEMEXP
MINDIS OQUERY PEAKSD PRUTXT RCMINI RS232I S6ZOOM SEMFAC
MINIMUM_* OUARR1 PEAKSR PS RCMIO RS232Q SAVER SEMFAL
MINPRP OUARR2 PEDIT PSDEFF RCMOP RS2BUF SAVER2 SEMFIL
MINX OUBISA PEXTR PSDEFI RCMSCL RS2CLR SAVE_IMA* SEMFLL
MINY OUBYSC PFERET PSDEFL RCMTCL RS2ENQ SAVE_VIE* SEMFOR
MINZ OUBYTE PFILT2 PSDEFS RD1CUR RS2ERR SBRAST SEMFRL
MKCHAR OUFPSA PFILT3 PSET RDALGN RS2MPT SCREEN SEMFRM
MKICHA OUPRGM PFILT4 PSHOW RDCURR RS2OFF SDNORD SEMFVD

SEMGDB	SEMPRO	SETVIE	SKETCH	SSNAME	SX11EX	TSRCOM	UIFMDA
SEMHAN	SEMRET	SETWAI	SMZERO	SSNUMB	SX11GC	TSTADP	UIFMDR
SEMHE2	SEMRGB	SET_ALL_*	SNYCNT	SSOKAY	SX11IN	TSTDAT	UIFMOB
SEMHE3	SEMRNG	SET_AUTO*	SNYFBA	SSOKAYF	SX11MC	TSTSRG	UIFMOE
SEMHE4	SEMROW	SET_BAND*	SNYGET	SSOOPS	SX11SC	TSTWAI	UIFMOP
SEMHEL	SEMRUN	SET_BORD*	SNYGRB	SSOUT	SX11VO	TURN	UIFMOQ
SEMHLN	SEMSEL	SET_CLIP*	SNYOFF	SSREAD	SXRCLR	TURN2	UIFMST
SEMICS	SEMSER	SET_COLO*	SNYPAR	SSREADF	SXRDRW	TURNKB	UIFMTY
SEMILC	SEMSIG	SET_CONT*	SNYPIN	SSREAD_F	SXRINS	TVMANU	UIFNO
SEMIN2	SEMSLG	SET_CURS*	SOFTKY	SSREVS	SXROUT	TWIST	UIFPAU
SEMIN3	SEMSOL	SET_DISP*	SOLID	SSSCAN	SXRSIZ	TXT	UIFPCO
SEMIN4	SEMSOP	SET_EMIS*	SOLID2	SSSEND	SXRUPD	TYPE	UIFPCR
SEMIN5	SEMSOU	SET_IMAG*	SOLID3	SSSENDF	SXSCAL	TZNAME	UIFPDE
SEMINI	SEMSY2	SET_LIGH*	SOLID4	SSSEND_F	SXVIEW	UIFAPE	UIFPH2
SEMINP	SEMSYN	SET_MATE*	SOURCE_L*	SSSTAT	SYNFIR	UIFCAD	UIFPHI
SEMINT	SEMSZZ	SET_MENU*	SPAWN	SSSTATF	SYNHST	UIFCBO	UIFPMA
SEMINX	SEMTAB	SET_MOUS*	SPINFT	SSSTAT_F	SYNIT	UIFCCC	UIFPNA
SEMIOB	SEMTBK	SET_OVER*	SPLER	SSSTOP	SYNLIV	UIFCCO	UIFPPO
SEMIOD	SEMTBS	SET_SING*	SPLIN1	SSSTOPF	SYNLOD	UIFCCY	UIFPSH
SEMIOE	SEMTCR	SET_SLID*	SPLINE	SSTELL	SYNMAG	UIFCDO	UIFPSI
SEMJMP	SEMTCU	SET_SMOO*	SPNECK	SSTRAP	SYNMAP	UIFCDR	UIFPTR
SEMKB1	SEMTEX	SET_WIRE*	SPROCED	SSVERS	SYNMOV	UIFCFC	UIFREA
SEMKB2	SEMTFC	SGI3DV	SPRTNE	SSWBUF	SYNROW	UIFCHI	UIFSAC
SEMKTX	SEMTFL	SGIARC	SPSORT	SSWHO	SYNSWP	UIFCIC	UIFSAD
SEMLA2	SEMTIT	SGIARZ	SQLINT	SSWHO_F	SYNXFR	UIFCIN	UIFSAS
SEMLA3	SEMTLF	SGIGHW	SRCM2	STACK	SYSTEM	UIFCOF	UIFSAV
SEMLA4	SEMTNV	SGIHAN	SRCM3	STAREG	SYS_ERRL*	UIFCOM	UIFSAW
SEMLAB	SEMTOU	SGINFA	SRCM30	START_EV*	SYS_NERR	UIFCRO	UIFSAZ
SEMLA2	SEMTOU_F	SHEET	SRDWRT	STLACL	S_BIOS	UIFCTC	UIFSCS
SEMLA3	SEMTPS	SHEET2	SRSCAL	STLAOK	S_CRIT	UIFCTE	UIFSSU
SEMLA4	SEMTRV	SHEET3	SRSGOP	STLAXS	TAPE	UIFDAC	UIFSTA
SEMLAY	SEMTXT	SHEET4	SRSGOR	STLCPY	TERSIZ	UIFDLO	UIFTAP
SEMLDB	SEMTYP	SHORT	SRSRDP	STLFIR	TEXTU1	UIFDQU	UIFTAS
SEMLIB	SEMUIS	SHOW	SRSRDR	STLFRD	TEXTU2	UIFDRE	UIFTCO
SEMLIN	SEMULN	SHOW0	SRSSLR	STLHST	TEXTU3	UIFDRF	UIFTCP
SEMLNF	SEMUNW	SHOW1	SRSWRR	STLMAG	THERML	UIFDSA	UIFTDE
SEMLOG	SEMUPP	SHOW2	SRSWUR	STLMAP	THRES2	UIFEAC	UIFTDR
SEMLOO	SEMUSA	SHOW22	SSCAN	STLMOV	THRESH	UIFECK	UIFTEX
SEMLU	SEMVW	SHOW3	SSCASE	STLSTT	TIFASC	UIFECO	UIFTIN
SEMLUT	SEMWAI	SHOW4	SSDEAD	STOLUT	TIFLON	UIFECR	UIFTLE
SEMMAC	SEMXA1	SHOW5	SSDEADF	STONAM	TIME	UIFED2	UIFTNU
SEMMED	SEMXIT	SHOW6	SSDONE	STOPDI	TIMER	UIFEDE	UIFTOU
SEMMEM	SEMXPL	SHOW7	SSEVEN	STOP_EVE*	TIMEZONE	UIFEDR	UIFTSE
SEMMON	SEMZZZ	SHOW8	SSEVENF	STORE	TIMSTR	UIFENA	UIFTTA
SEMMOR	SEPART	SHOW9	SSEXIT	STRING_T*	TOHEX	UIFEPO	UIFTVA
SEMNXT	SETBIT	SHOWLF	SSEXITF	STRIP_ZE*	TOHEXB	UIFESI	UIFTWP
SEMOP2	SETCH	SHOWNL	SSFIND	STRTDI	TOHEXL	UIFIAF	UISBCP
SEMOP3	SETCHE	SHOWNO	SSGEN	SURVEY	TOHEXW	UIFIBA	UISCPO
SEMOPN	SETDISPL*	SHOWSY	SSGO	SUSP	TOKEN	UIFIBE	UISDCU
SEMPAG	SETDRV	SHOW_CLI*	SSGONE	SWREAD	TPMESS	UIFIGO	UISROV
SEMPDB	SETLUT	SHOW_LIG*	SSHEET	SWTAST	TRACK_IN*	UIFIJU	UISWBS
SEMPIX	SETPPR	SHUTDOWN*	SSINTR	SX11CE	TRACK_ME*	UIFINI	UISWIP
SEMPLI	SETSCA	SHUT_FIL*	SSINTRF	SX11CR	TRACK_SL*	UIFINP	UISWOV
SEMPPN	SETUCD	SIGNGAM	SSLOOK	SX11CT	TRAIN	UIFITE	UIXCDH
SEMPRG	SETUP_WI*	SIZEM	SSMDI	SX11CU	TRNREG	UIFMAC	UIXCMP
SEMPRI	SETVAR	SKEAST	SSMDIF	SX11DX	TSEQAV	UIFMD2	UIXCO0

UIXCO1	UIXXDC	VFCPRT	VMSK24	WINCOL	XSSHME
UIXCO2	UIXXFI	VFCROW	WAIINT	WINCRE	XSSTER
UIXCO3	UIXXLE	VFCSAM	WAITCALL*	WINDES	XSTYLE
UIXCO4	UIXXMC	VFCX11	WAITS	WINDOW	XSUMEN
UIXCO5	UIXXMP	VGAGET	WAITU	WINHID	XSWEVT
UIXCO6	UIXXPO	VGAINI	WAL1D	WINPOS	XSWRLT
UIXCO7	UIXXPU	VGALUT	WALSH	WINSHO	XWCIRC
UIXCO8	UIXXRM	VGAPUT	WALSH1	WINTEX	XWCURV
UIXCPO	UIXXTP	VGARDD	WALSH2	WPRINT	XWGRAP
UIXCPU	ULPDET	VGATXD	WALSH3	WRALGN	XWIBOX
UIXCRA	UNDER0	VGAWRD	WARP	WRICLI	XWIRES
UIXCSI	UNDFL	VGBIOS	WBOD32	WRILUT	XWLINE
UIXDRE	UNPACK	VGCON	WCBD32	WRITCB	XWLIST
UIXEIN	UNSETV	VGMASK	WDXBOX	WRITEC	XWPOIN
UIXEPO	UPPER	VGPGDN	WDXCLR	WRITE_DE*	XWREGI
UIXEQU	USER	VGPGUP	WDXCOL	WRITE_FI*	YESNO
UIXEVG	USLEEP	VGREDW	WDXCOW	WRITE_IM*	YMOD
UIXEVP	USTCAT	VIDEO	WDXCRE	WRITE_LU*	ZCHOPF
UIXEVR	USTCMP	VIDEOL	WDXDES	WRITE_ME*	ZERO_DEV*
UIXFBF	USTCPY	VIDEOR	WDXFNS	WRITE_OV*	ZERO_FIL*
UIXGCL	USTLEN	VIEW	WDXFST	WRITE_SE*	ZERO_MEM*
UIXICA	USTNCA	VLCLIP	WDXFVS	WRITFD	ZERO_SEG*
UIXICM	USTNCP	VMCACT	WDXHID	WRITSE	ZFEXA
UIXIEC	USTNUL	VMCALL	WDXPOS	WRITSO	ZRDWRT
UIXINP	UVCELL	VMCBEG	WDXSHO	WRMODE	
UIXINT	UVDEVI	VMCHLT	WDXTEX	WRSTIG	
UIXIOM	UVELE1	VMCLOA	WDXXHL	WRTEPM	
UIXIPB	UVELE2	VMCPRG	WDXXVL	WRUMOD	
UIXIPE	UVEXEC	VMCVEC	WEIGHT	WTITLE	
UIXIPK	UVJUST	VMEDIN	WEP	X11DLN	
UIXIPP	UVMENU	VMEENA	WFSBOX	X11DTX	
UIXIRC	UVMOUS	VMEINI	WFSCLR	X11MAP	
UIXISC	UVPANE	VMERBB	WFSCNA	X11SET	
UIXISD	UVTEXT	VMERBC	WFSCOL	X11ULN	
UIXISE	UVUIF	VMERBI	WFSCRE	X12BA	
UIXMPC	V	VMERBL	WFSDES	XCF	
UIXMSI	VAL	VMERBO	WFSHID	XCM	
UIXOBS	VALBIT	VMERBT	WFSPOS	XCMBA	
UIXODA	VARINT	VMERLW	WFSSHO	XCNV	
UIXOIN	VARSET	VMERWD	WFSTEX	XSDDIS	
UIXPD2	VERTICAL*	VMEWBB	WGRAB	XSDMEN	
UIXPDE	VERT_SCA*	VMEWBC	WGRABL	XSDNOT	
UIXPDR	VFC	VMEWBI	WGRABV	XSDSTR	
UIXPIN	VFC2	VMEWBL	WHOBOX	XSDTER	
UIXPSP	VFCACQ	VMEWBO	WHOCLR	XSDVIR	
UIXROP	VFCAREA	VMEWBT	WHOCNA	XSFUPD	
UIXRSR	VFCBOF	VMEWBW	WHOCOL	XSGDEF	
UIXSAD	VFCBON	VMEWCB	WHOCRE	XSHICU	
UIXSCL	VFCDUN	VMEWCC	WHODES	XSLCOL	
UIXSER	VFCGRB	VMEWCI	WHOHID	XSMMEN	
UIXSIN	VFCKEY	VMEWLW	WHOPOS	XSMSTR	
UIXSOF	VFCMESS	VMEWWD	WHOSHO	XSMVIR	
UIXTDC	VFCMOD	VME_BASE	WHOTEX	XSNDES	
UIXTSI	VFCMODE	VME_BUS	WINBOX	XSNUNK	
UIXWIN	VFCOFF	VME_CATC*	WINCLR	XSQSTR	
UIXXCP	VFCPORT	VMSK16	WINCNA	XSSHCU	

Long reserved names
==========

ALLOCATE_BUFFER
ALLOCATE_MEMORY
ALLOCATE_SEGMENT
AMBIENT_BLUE
AMBIENT_BLUE_MOVED
AMBIENT_GREEN
AMBIENT_GREEN_MOVED
AMBIENT_RED
AMBIENT_RED_MOVED
ASPECT_RATIO
AUTO_CENTRING_ON
BACK_PLANE_PERCENTAGE
BACK_SECTION_PLANE_MOVED
BAND_COARSENESS
BAND_COARSENESS_MOVED
BBCALLBACK
BORDER_ON
BUFENDTAB
CACHECOLORMASK
CENTRE_VIEW
CHARACTER_TO_PIXEL
CHARACTER_TO_PIXEL_COUNT
CHECK_FOR_CANCEL
CLEAR_WINDOW
CLIPPING_ON
CLIP_X_MAXIMUM
CLIP_X_MINIMUM
CLIP_Y_MAXIMUM
CLIP_Y_MINIMUM
CLIP_Z_MAXIMUM
CLIP_Z_MINIMUM
CLOSE_DISPLAY_WINDOW
CLOSE_FILE
CLOSE_MEMORY
CLOSE_WINDOW
CONVERT_SLIDER_VALUE
COPY_MEMORY
CREATE_CACHE
CREATE_FILE
CREATE_SEGMENT_TABLE
CURSOR_ON
CURSOR_X_MOVED
CURSOR_Y_MOVED
CURSOR_Z_MOVED
DELETE_FILE
DISPLAY_ASSIGN
DRAWING_MODE
DRAW_3D_CURSOR
DRAW_BANDED_FACET
DRAW_BANDED_QUADRILATERAL
DRAW_BANDED_TRIANGLE
DRAW_BORDER
DRAW_CIRCLE
DRAW_CORNER_FACET
DRAW_CURSOR
DRAW_MENU
DRAW_MIDDLE_FACET
DRAW_MODEL
DRAW_NOTE
DRAW_PLANE_FACET
DRAW_RECTANGLE
DRAW_SHADED_HEXAGON
DRAW_SHADED_PENTAGON
DRAW_SHADED_QUADRILATERAL
DRAW_SHADED_TRIANGLE
DRAW_SIDE_FACET
DRAW_SLIDER
DRAW_SLIDER_BOX
DRAW_SLIDER_VALUE_AND_POINTER
DRAW_STRING
DRAW_WINDOW
DUPLICATE_STRING
EDITCALLBACK
ELEVATION_ANGLE
EYE_AZIMUTH_MOVED
EYE_DISTANCE
EYE_ELEVATION_MOVED
FILL_IMAGE_BLOCK
FILL_IMAGE_PIXEL
FILL_MEMORY_BLOCK
FILL_MEMORY_PIXEL
FILL_OVERLAY_BLOCK
FILL_OVERLAY_PIXEL
FLUSHDISPLAYFILE
FLUSH_DEVICE
FLUSH_SEGMENT
FLUSH_SEGMENT_TABLE
FORCE_UPDATE
FORM_TO_BYTES
FOV_ANGLE
FOV_MOVED
FREEDISPLAY
FREEFRAMES
FREE_CACHE
FREE_DOS_FILE_HANDLES
FREE_MEMORY
FREE_SEGMENT
FREE_SEGMENT_TABLE
FRONT_PLANE_PERCENTAGE
FRONT_SECTION_PLANE_MOVED
GETDISPLAYFILE
GETDISPLAYINT
GETDISPLAYSTRING
GET_BANDED_VERTEX
GET_BASES
GET_CHARACTER_SIZE

GET_COLOURED_VERTEX
GET_DISPLAY_BUFFER_SIZE
GET_DISPLAY_WINDOW_ID
GET_DISPLAY_WINDOW_MODE
GET_DISPLAY_WINDOW_POSITION
GET_DISPLAY_WINDOW_SIZE
GET_FONT_SIZE
GET_IMAGE_LUT
GET_IMAGE_LUT_SIZE
GET_INTERFACE_WINDOW_ID
GET_OVERLAY_COLOURS
GET_OVERLAY_DISPLAY_MASK
GET_OVERLAY_LUT
GET_OVERLAY_LUT_SIZE
GET_OVERLAY_WRITE_MASK
GET_SHADED_VERTEX
GET_VERTEX_POSITION
GID_TO_WINDOW_INDEX
GMCAMERAF
GMENQUIRE
GMENQUIREF
GMFILTERF
GMGETDEFAULT
GMGETDEFAULTF
GMLUTVIEW
GMLUTVIEWF
GMOUTLUTF
GMOVERLAY
GMOVERLAYF
GMOVERREAD
GMOVERREADF
GMOVERREADPACKED
GMOVERVIEW
GMOVERVIEWF
GMOVERWRITE
GMOVERWRITEPACKED
GMSETDEFAULT
GMSETDEFAULTF
GMSETGAIN
GMSETGAINF
GMSETOFFSET
GMSETOFFSETF
GMSTANDARD
GMSTANDARDF
GMSUPPORT
GRAB_DOS_FILE_HANDLES
HANDLE_MOUSE_HIT
HIDE_CLIPPING
HIDE_LIGHTING
HOURGLASS_CURSOR
HWREG_LOCK
INTERPOLATE_VERTEX
INTERRUPTREGISTERCALLBACK
IP12_BASE
IP_NUMBER
KEEPINVENT
LIGHT_1_AZIMUTH_MOVED
LIGHT_1_BLUE_MOVED
LIGHT_1_ELEVATION_MOVED
LIGHT_1_GREEN_MOVED
LIGHT_1_RED_MOVED
LIGHT_2_AZIMUTH_MOVED
LIGHT_2_BLUE_MOVED
LIGHT_2_ELEVATION_MOVED
LIGHT_2_GREEN_MOVED
LIGHT_2_RED_MOVED
LIGHT_3_AZIMUTH_MOVED
LIGHT_3_BLUE_MOVED
LIGHT_3_ELEVATION_MOVED
LIGHT_3_GREEN_MOVED
LIGHT_3_RED_MOVED
LIGHT_BLUE
LIGHT_ELEVATION
LIGHT_GREEN
LIGHT_RED
LIGHT_ROUND
LINE_3D_TO_PIXEL
LINE_TO_PIXEL
LOAD_SEGMENT
MAKE_FILE_NAME
MAXIMUM_X_MOVED
MAXIMUM_Y_MOVED
MAXIMUM_Z_MOVED
MAX_BAND_COARSENESS
MAX_MODEL_COARSENESS
MINIMUM_X_MOVED
MINIMUM_Y_MOVED
MINIMUM_Z_MOVED
MODEL_COARSENESS
MODEL_COARSENESS_MOVED
MONITORLOCK
MOUSE_TO_PIXEL
MOVE_SLIDER
OPEN_DISPLAY_WINDOW
OPEN_FILE
OPEN_MEMORY
OPEN_NEW_FILE
OPEN_OLD_FILE
OPEN_SCRATCH_FILE
OPEN_WINDOW
OUTPUT_PICTURE_NO
POSITION_MENU
POSITION_SLIDER
QUERYCALLBACK
QUERY_CACHE
QUERY_CACHE_LIMITS
QUERY_DEVICE
QUERY_FILE
QUERY_MEMORY
QUERY_SEGMENT
READ_DEVICE
READ_DEVICE_DATA

READ_FILE
READ_IMAGE_BLOCK
READ_IMAGE_PIXEL
READ_MEMORY_BLOCK
READ_MEMORY_PIXEL
READ_OVERLAY_BLOCK
READ_OVERLAY_PIXEL
READ_SEGMENT
READ_SEGMENT_DATA
REDRAW_DISPLAY_BLOCK
REDRAW_DISPLAY_PIXEL
REDRAW_DISPLAY_WINDOW
REDRAW_INTERFACE_WINDOW
REDRAW_MODE
REDRAW_SSHEET_WINDOW
RESIZE_DEVICE
RESIZE_FILE
RESIZE_MEMORY
RESTORE_VIEW_PARAMETERS
ROUND_ANGLE
SAVE_IMAGE
SAVE_VIEW_PARAMETERS
SECURECALLBACK
SELECT_CENTRE_VIEW
SELECT_QUIT
SELECT_RESTORE_VIEW
SELECT_SAVE_IMAGE
SELECT_SAVE_VIEW
SETDISPLAYINT
SETDISPLAYSTRING
SETUP_WINDOWS
SET_ALL_OVERLAYS
SET_AUTO_CENTRING_OFF
SET_AUTO_CENTRING_ON
SET_BAND_DATA
SET_BORDER_OFF
SET_BORDER_ON
SET_CLIPPING_OFF
SET_CLIPPING_ON
SET_COLOUR_CONTOUR_RENDERING
SET_COLOUR_MAP_RENDERING
SET_CONTINUOUS_REDRAW
SET_CURSOR_OFF
SET_CURSOR_ON
SET_CURSOR_VISIBILITY
SET_DISPLAY_WINDOW_POSITION
SET_DISPLAY_WINDOW_SIZE
SET_EMISSION_RENDERING
SET_IMAGE_LUT
SET_LIGHTING_MODEL
SET_LIGHT_SOURCES
SET_MATERIALS
SET_MENU_SELECTED_LINES
SET_MOUSE_POSITION
SET_OVERLAY
SET_OVERLAY_COLOURS
SET_OVERLAY_DISPLAY_MASK
SET_OVERLAY_LUT
SET_OVERLAY_WRITE_MASK
SET_SINGLE_REDRAW
SET_SLIDER_POSITION
SET_SLIDER_VALUES
SET_SMOOTH_RENDERING
SET_WIREFRAME_RENDERING
SHOW_CLIPPING
SHOW_LIGHTING
SHUTDOWN_WINDOWS
SHUT_FILE
SOURCE_LP_NO
START_EVENT_QUEUE
STOP_EVENT_QUEUE
STRING_TO_PIXEL
STRING_TO_PIXEL_COUNT
STRIP_ZEROS
SYS_ERRLIST
TRACK_INPUT
TRACK_MENU
TRACK_SLIDER
VERTICAL_SCALE
VERT_SCALE_MOVED
VME_CATCH
WAITCALLBACK
WRITE_DEVICE
WRITE_DEVICE_DATA
WRITE_FILE
WRITE_IMAGE_BLOCK
WRITE_IMAGE_PIXEL
WRITE_LUT
WRITE_MEMORY_BLOCK
WRITE_MEMORY_PIXEL
WRITE_OVERLAY_BLOCK
WRITE_OVERLAY_PIXEL
WRITE_SEGMENT
WRITE_SEGMENT_DATA
ZERO_DEVICE
ZERO_FILE
ZERO_MEMORY
ZERO_SEGMENT